VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Temná inteligence a malware

Dark Intelligence in Malware

Bc. Petr Řepa

Diplomová práce

Vedoucí práce: prof. Ing. Ivan Zelinka, Ph.D.

Ostrava, 2021

Abstrakt

Cílem mé diplomové práce je vytvoření malwaru, který předvádí a využívá prvky hejnové inteligence s použitím prostředků a služeb temného webu. V této práci se pokusím takový malware experimentálně naprogramovat a vyzkoušet vlastnosti přístupu s důrazem na vlastnosti temného webu. V teoretické části popisuji funkci sítě Tor, její základní pojmy, na kterých je postaveno výsledné řešení praktické části. Dále zmíním nejznámější algoritmy hejnové inteligence, jimiž jsem se inspiroval při tvorbě výsledného malwaru. V další části zdůvodním, proč je jazyk Python vhodný pro moji práci. V praktické části popíšu jednotlivé komponenty, ze kterých se výsledné řešení skládá. Závěrem předvedu praktickou ukázku spuštění malwaru od nákazy, přes distribuci nového uzlu až po vykonání vzdáleného payloadu.

Klíčová slova

Malware; Hejnová inteligence; Tor; Darkweb; Payload;

Abstract

The aim of this diploma thesis is to create a malware that will demonstrate the principles of swarm intelligence while simultaneously using the resources and services of the dark web. In this thesis, I will try to implement the malware with emphasis on the properties of the dark web. The theoretical part of the thesis describes the functionality of the Tor network and the basic terminology surrounding the network, both of which are the basis which the final implementation is built on. Furthermore, I will also mention the most famous swarm intelligence algorithms that the implementation was inspired by. In the following part of the thesis, I will explain the motivation behind choosing Python as the primary programming language to implement the malware. In the practical part of the thesis, the respective components of the resulting solution will be introduced. Finally, I will demonstrate the functionality of the malware, beginning with its startup, continuing with the distribution of a new node and eventually executing remote payload.

Keywords

Malware; Swarm Intelligence; Tor; Darkweb; Payload;

Poděkování

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla. Především pak vedoucímu mé diplomové práce prof. Ing. Ivanu Zelinkovi, Ph.D.

Obsah

Seznam použitých zkratek a symbolů

AES	–	Advanced Encryption Standard
C2	–	Command and Control
C&C	–	Command and Control
TCP/IP	–	Transmission Control Protocol/Internet Protocol
HTTP(S)	–	Hypertext Transfer Protocol (Secure)
TOR	–	The Onion Routing
SSH	–	Secure Shell
RSA	–	Rivest, Shamir, Adleman

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod a cíl práce

1.1 Úvod

Za posledních 30 let se svět počítačů výrazně změnil. Díky tomu, že jsou počítače daleko rychlejší, mohou programátoři využívat technologie, které byly dříve nemyslitelné. Bavíme se o nástrojích, které pomáhají samotným programátorům, jako jsou komplexní vývojové prostředí, tak samotné programovací jazyky, které se svojí mírou abstrakce začínají přibližovat lidskému myšlení. Na druhou stranu tento rapidní vývoj počítačů usnadňuje také práci tvůrcům malwaru.

Před několika lety byla tvorba malwaru velmi náročnou oblastí, do které se pouštěli jen velmi zkušení programátoři. Tvůrci malwaru tvořili viry a jiné škodlivé kódy spíše pro ukázku svých programátorských schopností s cílem ničit obsahy pevných disků. Dnes je hlavní motivací útočníka především zisk.

Díky rychlosti počítačů už tvůrce malwaru nepotřebuje využívat nízkoúrovňový programovací jazyk assembler, popřípadě C, ale pro jeho tvorbu bohatě postačí například Python, který sice nedosahuje rychlosti assembleru, nicméně jeho uživatelská přívětivost je daleko větší. Při použití jazyka jako je právě Python, může tvorba malwaru, který např. zašifruje uživatelská data a vyžaduje platbu pomocí virtuální měny, být otázkou několika hodin.

S příchodem internetu se také změnil celkový koncept malwaru. Velká část dnešních malwarů využívá architektury C&C. Pomocí tohoto přístupu může útočník na dálku ovládat nakažené počítače. Majitel řidícího serveru jednoduše odesílá příkazy na nakažené počítače a ty je vykonávají. Zde však přichází ta kritická část centrálního řízení malwaru. Ve chvíli, kdy se bezpečnostním složkám podaří odstranit tento řídicí server, nakažený počítač už nemá jak dál získávat nové příkazy.

Zároveň díky internetu po sobě útočníci většinou zanechávají nějakou stopu; co kdyby ale existoval malware, který by chránil anonymitu autora malwaru a zároveň by neměl jen jeden řídicí server?

V této práci bych rád vytvořil malware, který využívá vysokoúrovňový programovací jazyk. Je velmi těžko vystopovatelný a zároveň ho nelze jen tak vypnout.

1.2 Cíl práce

Cílem mé diplomové práce je vytvoření malwaru, který předvádí a využívá prvky hejnové inteligence s použitím prostředků a služeb temného webu.

Práce je zaměřena na pochopení základních principů a možností spojení hejnové inteligence, malware a temného webu (dark webu).

V této práci se pokusím takový malware experimentálně naprogramovat a vyzkoušet vlastnosti takového přístupu s důrazem na vlastnosti temného webu. Navržený prototyp by měl být modulární, a to s podporou připojení budoucích modulů.

Kapitola 2

State Of Art

V devadesátých letech se většina virů a jiných škodlivých kódů tvořila v kompilovaných programovacích jazycích jako je Assembler, C, Pascal a další. Jednalo se především o počítačové viry, které měly za úkol mazat obsahy pevných disků, vytěžovat počítač či jinak škodit. Za tu dobu se počítače velmi změnily. Oproti devadesátým létům jsou procesory počítačů mnohonásobně rychlejší, kapacity operačních pamětí se pohybují v řádu jednotek až desítek gigabajtů a internet má dnes už v podstatě každý. Výrazně se také změnila oblast programovacích jazyků. Díky tomu, že se výpočetní výkon počítačů stal levnější než cena programátorů, mohou programátoři využívat programovací jazyky, ve kterých je vývoj daleko rychlejší a příjemnější. Kromě kompilovaných jazyků můžeme zvolit například jazyky, které využívají virtuální stroj jako je C# či Java nebo interpretované, mezi které řadíme například JavaScript. Ve zmíněných jazycích je ještě větší míra abstrakce a zápis kódu je tak blíže lidskému myšlení. Od roku 2008 se oblíbenost interpretovaného jazyka Python neustále zvyšovala a v roce 2020 se pak Python objevil na špičce ve statistice oblíbenosti programovacích jazyků magazínu IEEE Spectrum. [1]

Python se v oblasti počítačové bezpečnosti stal určitým standardem. Jeho uplatnění můžeme najít v jednoduchých skriptech dokazujících existenci nějaké zranitelnosti, automatizaci útoků popřípadě nástrojů pro vzdálené otevření shellu, neboli Remote Access Tool. Dvacet procent všech repositářů na webu github.com, které se orientují na zmíněné oblasti, využívají právě Python. [2]

Interpretované jazyky však mají jednu velkou výhodu oproti kompilovaným a tou je vykonávání kódu až za chodu. S příchodem automatizačního nástroje PowerShell začali tvůrci škodlivého kódu tuto vlastnost ve velké míře využívat. Kód napsaný v PowerShellu totiž můžeme předat interpretu powershell.exe a ihned ho vykonat, aniž bychom kód museli vkládat do samostatného souboru. Zde narážíme na podstatu fileless malwaru, tedy malwaru, který pro svůj běh nepotřebuje samostatný soubor. Samotný kód včetně cesty k interpretu pak vložíme přímo do registrů. Tím zajistíme, že škodlivý kód bude nenápadný a bude se automaticky spouštět při startu počítače. Od roku 2017 patří fileless malware mezi jednu z nejčastějších hrozeb. [3]

PowerShell je od roku 2016 dostupný také na platformách Mac OS X a Linux, nicméně zatím pouze jako balíček, který je potřeba doinstalovat. [4]

PowerShell sice není v základní instalaci linuxových distribucí a systému Mac OS X, to však neznamená, že se těmto dvěma platformám fileless malware vyhýbá.

Linux a Mac OS X mají totiž v základu interpret Pythonu a výše zmíněný příklad s PowerShellem lze obdobně aplikovat pomocí Pythonu. Rozdíl spočívá v tom, že místo registrů se využijí jiné možnosti operačního systému Mac OS X.

V roce 2017 na serveru Fortinet vyšla analýza, která popisuje nákazu pomocí dokumentu Word, který obsahuje škodlivý VBA kód. Word dokument zde funguje jako dropper, který otestuje, zda je program Word spuštěný pod Windows nebo Mac OS X a na základě toho spustí payload buď v PowerShellu nebo v Pythonu. V případě Pythonu se spouští kód, který otevře do počítače zadní vrátka pro vykonávání vzdáleného kódu útočníka. [5]

Kromě výše zmíněného příkladu se začínají objevovat vzorky malwaru, které jsou napsány v Python a slouží pro platformu Windows. Takový malware si kromě samotného kódu musí s sebou nést i kompletní interpret jazyka Python. K tomuto účelu slouží například nástroj PyInstaller, který z jednoho nebo i více Python skriptů udělá jeden spustitelný soubor.

Tor je počítačová síť, která se využívá pro ochranu identity uživatele a zabraňuje možnému odposlouchávání prostřednictvím man-in-the-middle útoků. Tyto vlastnosti mohou využívat jak občané různých totalitních režimů, kteří nemají svobodný přístup k internetu a informacím, ale také prodejci různého ilegálního obsahu, tvůrci malwaru a hackeři.

Malware šířící se po síti Tor není ničím novým. V srpnu roku 2013 se náhle zvýšil počet uživatelů Tor z necelého milionu na pět milionů, za tímto zvýšením stál malware Mevade, který začal používat Tor jako jednu z komponent. [6]

Společnost Trend Micro v roce 2014 vydala report, ve kterém popsala tehdejší nejčastější hrozby, které využívaly síť Tor pro šifrovanou komunikaci. Sem patřil i malware Mevade. Tento malware měl několik variant. V první variantě se malware do počítače dostal jako trojský kůň po instalaci falešného Adobe Flash Playeru. Trojan poté stáhl a nainstaloval adware včetně různých toolbarů do prohlížečů, pomocí kterého obětem zobrazoval reklamy, na kterých profitoval. V prvních verzích využíval HTTP a SSH pro komunikaci s C2 serverem. Kromě adwaru instaloval na počítače také backdoor, který mohl sloužit pro kradení dat. Tato verze je označována jako BKDR_MEVADE.A. Varianta, která využívá komponentu Toru, je označována jako BKDR_MEVADE.B a BKDR_MEVADE.C. Tuto komponentu využívala pro komunikaci s C2 serverem jako náhradu za předchozí HTTP a SSH komunikaci. [7]

Tato C&C kampaň je jedna z největších botnet kampaní dodnes. Od té doby se situace na poli malwaru výrazně změnila. Kyberzločinci se začali ve velké míře zaměřovat na ransomware a dnes už v podstatě každý ransomware využívá síť Tor pro platbu za výkupné. Díky využití sítě Tor jsou zločinci velice těžko dohledatelní, protože veškerý přenos je šifrovaný a odchytávání paketů a následná analýza je tak velmi složitá.[8]

V roce 2020 objevili výzkumníci společnosti ESET trojského koně, který na počítači oběti těžil kryptoměny, snažil se získat přístup ke kryptoměnovým peněženkám a při zkopírování veřejné adresy kryptoměnové peněženky do schránky ji vyměnil za svou vlastní. Ve chvíli, kdy uživatel chtěl odeslat např. bitcoiny na jinou adresu a tuto adresu vložil ze schránky do pole pro adresu příjemce a nevšiml si, že je adresa jiná, poslal bitcoiny na adresu tvůrců malwaru. Tento malware cílí především na uživatele českého a slovenského internetu. Uživatel si nakazil počítač sám ve chvíli, kdy stáhl infikovaný soubor z webového úložiště Ulož.to. Výzkumníci ESET pojmenovali malware KryptoCibule. [9]

Tyto a další malwary však mají jednu nevýhodu a tou je C2 server. Ve chvíli, kdy se podaří odstranit tyto centrální jednotky, nakažené počítače již nemají jak dostávat nové pokyny.

Tento problém může řešit použití hejnové inteligence.

Hejnová inteligence je typ umělé inteligence, která se snaží napodobit chování hejna v přírodě. Samotní jedinci hejna si se svými sousedy vyměňují zprávy bez nutnosti nějaké centrální jednotky. Tímto způsobem může hejno aktivně vyhledávat potravu, aniž by vědělo, kde je daný cíl.

Mravenci v přírodě značkují svoje cesty pomocí feromonů, díky kterým mohou přitahovat další mravence. Čím více mravenců po dané cestě putuje, tím je pach feromonů silnější a cesta přitahuje další jedince. Tímto způsobem se označí správné cesty a naopak cesty s malým množstvím feromonů postupně zanikají. Na základě tohoto biologického jevu byl vytvořen algoritmus Ant Colony Optimization.

Počítačový virus využívá biologicky inspirovaného algoritmu pro průchod souborovým systémem. Algoritmus využívá hodnotící funkci, tzv. fitness funkci, pro ohodnocení každého souboru. Klasický počítačový virus se většinou snaží určitým způsobem škodit. Virus založený na ACO však používá soubory pouze pro skok na další soubor. Při skoku na nový soubor se virus z původního souboru smaže. Kvalita každého souboru se hodnotí na základě předem nastavených vlastností. Lze jmenovat například velikost souboru, typ souboru, popřípadě čas vytvoření.

Tímto tématem se již nějakou dobu zabývá tým okolo profesora Ivana Zelinky. Skupina se snaží chování takového viru simulovat, aby v budoucnu bylo možné na podobné hrozby reagovat nebo se jim úplně předcházet. [10]

Kapitola 3

The Onion Routing

Web můžeme v základě rozdělit na dvě části. Na část na povrchu (surface) a na část pod povrchem (deep). Stránky na povrchu jsou dohledatelné pomocí internetového vyhledávače jako je například DuckDuckGo. Tyto stránky jsou takzvaně indexované. Tuto část internetu nazýváme surface web. [11] [12]

Stránky, které nejdou takto jednoduše dohledat, můžeme označit jako "pod povrchem". Sem patří například facebooková konverzace, emailová pošta, uzavřená internetová fóra, ale i nelegální činnost. Do této části webu spadá právě dark web. Na dark webu můžeme najít různá ilegální tržiště, ale i speciální verze známých webů jako je Facebook, CNN, popřípadě vyhledávače DuckDuckGo. Dále je potřeba říct, že během vzniku internetu vznikaly sítě, do kterých se připojíte jen pomocí specializovaného softwaru. Tyto sítě se označují jako darknety. Nejznámější darknetem je síť Tor. Na dark web se přistupuje právě pomocí různých darknetů. Kromě darknetu Tor existuje také např. Freenet a I2P. [11] [12]

Ve své práci využiji Tor pro komunikaci jednotlivých klientů botnetu. Tato kapitola popisuje základní pojmy, které jsou nezbytné pro pochopení funkcionality výsledného řešení.

3.1 Tor

Tor je program, který běží na pozadí počítače a stará se o větší bezpečnost uživatele na internetu. Aplikace Tor uživatele chrání tím, že jeho komunikaci přeposílá přes síť složenou z počítačů dobrovolníků. Tato síť se taktéž označuje slovem Tor. Název Tor byl původně odvozen od The Onion Routing.

Software, který se využívá pro komunikaci v Tor síti se také někdy označuje jako Core Tor.

Pro přístup do sítě Tor uživatele nejčastěji využívají Tor Browser, což je prohlížeč založený na Firefoxu, který přidává speciální nastavení pro maximální soukromí uživatelů [13] [14]

3.2 TOR

The Onion Routing neboli cibulové směrování, značí způsob, kterým jsou data po cestě zašifrována. Pokud odešleme nějaká data přes síť Tor, jsou data zašifrována po vrstvách (slupkách), tak jako v případě cibule. Při odesílání dat si náš počítač náhodně vybere počítače tří dobrovolníků, přes které data proputují. Počítač dobrovolníka v síti Tor se označuje jako relay. Seznam těchto relay je veřejně dostupný, stejně tak jejich veřejné klíče. Data jsou při přenosu zašifrována právě veřejnými klíči těchto dobrovolníků. Díky tomu může každý relay dešifrovat jen tu vrstvu, od které má privátní klíč. Data jsou tedy na začátku cesty zašifrována třikrát. Kromě zašifrovaných dat každý relay ví, jakému dalšímu relay má data přeposlat. Šifrovaná data se na každém relay zbaví jedné vrstvy. Nejbezpečnější jsou data při odesílání prvnímu uzlu, který označujeme jako Guard. Prostřední uzel pouze ví, odkud data přišla a komu je má odeslat. Uzel označujeme jako Middle Relay. Poslední uzel označujeme jako Exit Node. Exit Node má již data v čitelné podobně a odesílá je na dotazovaný server. Exit Node sice ví, jaký je obsah dat, ale neví, odkud přišla. Zároveň Guard ví, odkud data přišla, ale neví, jaký je jejich obsah. [15]

3.3 Obvod (circuit)

Je cesta v síti Tor, která je složena z náhodně vybraných uzlů. Většina obvodu se skládá ze tří uzlů. Jako první uzel může být buď Guard nebo Bridge. Prostřední uzel se označuje jako Middle Relay a poslední jako Exit Node. Toto složení uzlů však platí jen v případě, že výsledná stránka je na surface webu. Tedy vystupujeme ze sítě Tor. Pokud přistupujeme na adresu s TLD .onion, Exit Node se vynechá. [16]

3.4 Onion Service

Dříve označována jako Onion Hidden Service, je služba jako například webová stránka, která je přístupná pouze ze sítě Tor.

Oproti klasickým webovým stránkám mají Onion Services jednu důležitou výhodu a tou je soukromí. Onion Service chrání uživatele dané služby, ale i jejich provozovatele.

Onion Services využívají klasicky šest uzlů pro spojení Tor obvodu. Tři uzly si zvolí klient a další tři služba.

Dále je také potřeba říct, že onion služby využívají speciální URL adresy. Tyto adresy jsou veřejným klíčem dané služby.

Existuje také Single Onion Service, což je služba, která sama o sobě nepotřebuje anonymitu, nicméně chce zajistit anonymitu svým uživatelům. Místo klasických šesti skoků využívá pouze tři a to pro jejich klienty. [17]

3.5 Zabezpečení

Ve chvíli, kdy uživatel přistoupí na adresu onion služby, tak díky end-to-end šifrování si může být jistý, že komunikuje pouze a jen s tou danou službou a komunikaci neodposlouchává nikdy jiný.

Šifrování funguje podobně jako HTTPS u klasických webových stránek, nicméně nemusíme se zde spoléhat na certifikační autoritu. Některé weby jako např. Facebook či ProtonMail, které jsou dostupné přes síť Tor, však obsahují také HTTPS, nepotřebují se totiž nějak skrývat, tak jako by to bylo v případě nějakého ilegálního obchodu. V tomto případě tak služba poskytuje uživateli anonymitu a dvojitou metodu šifrování dat. [17] [18] [19]

3.6 NAT

Pro přístup na internet z univerzitní sítě nebo z práce může být problém s firewallem. Firewall totiž může některé porty blokovat. Tor toto omezení obchází, a to tak, že navazuje pouze výstupní spojení. [20]

3.7 Skrývání IP adres

Na klasickém internetu se pro připojení na nějakou webovou stránku využívají IP adresy. Každá webová stránka musí mít svoji veřejnou IP adresu, na kterou se uživatelův počítač následně připojí. Onion služby fungují jako překryvná síť nad nejvyšší vrstvou modelu TCP/IP. Na této úrovni se IP adresy nepoužívají. Díky tomu je IP adresa služby chráněna. [20]

3.8 Rendezvous

Pro spojení mezi službou a klientem se využíval takzvaný rendezvous point.

3.8.1 Navázání spojení mezi klientem a službou

Pokud někdo vytvoří novou onion službu, tak se na pozadí udělá několik operací.

V první řadě samotná služba kontaktuje několik Tor relay, kterým oznámí, že naváže dlouhodobé spojení a požaduje, aby se dané relay chovaly jako tzv. introduction pointy. Veškeré spojení je anonymizováno, takže nedojde k odhalení identity služby daným relay uzlům. Služba bude dostupná pouze z těchto tří relay uzlů. Spuštěná služba se celou dobu bude tvářit jako kterýkoliv jiný uživatel sítě Tor. [20]

Ve chvíli, kdy je navázáno spojení s introduction pointy, musíme našim budoucím klientům zprostředkovat cestu, aby se k nim dostali a mohli tak komunikovat s naší službou. Z tohoto důvodu musí služba vytvořit Onion Service Descriptor, což je záznam, který obsahuje seznam introduction pointů včetně ověřovacích klíčů. Tento deskriptor podepíše samotná služba svým privátním klíčem.

Privátní klíč je částí veřejného klíče, který je zakódován v adrese onion služby. Takto podepsaný deskriptor nahraje onion služba do distribuované hašovací funkce, o kterou se stará directory server. Distribuovaná hašovací funkce je také částí Tor sítě, takže se k ní klient dostane. Samotné nahrání také používá anonymizovaný obvod, takže opět nedojde ke zveřejnění polohy. V tuto chvíli je služba již zveřejněná. [20]

Nyní musíme předat onion adresu našemu klientovi. Po zadání onion adresy služby do adresního řádku vyhledávače a stisknutí klávesy enter si Tor Browser vyžádá podepsaný deskriptor od distribuované hašovací funkce. Po získání podepsaného deskriptoru se ověří jeho podpis pomocí veřejného klíče, který je zakódovaný v onion adrese. Díky ověřenému podpisu si klient může být jistý, že komunikuje se službou, která je dostupná přes danou onion adresu. Onion adresa je totiž také kryptografický klíč, pomocí kterého můžeme ověřit autenticitu deskriptoru. [20]

V tuto chvíli si klient vybere Tor relay a naváže s ním spojení. Toto relay zároveň požádá, aby vystupovalo jako rendezvous point a předá mu tajný jednorázový řetězec. Tento jednorázový řetězec se použije později jako část ověřovacího procesu rendezvous pointu. [20]

Poté klient kontaktuje jeden z introduction pointu, kterému oznámí, že se daný klient chce spojit s danou službou. Tomuto introduction pointu klient také musí předat informace, jako je adresa rendezvous pointu a tajný jednorázový řetězec. Introduction point tyto detaily předá službě, která následně spustí několik ověřovacích procesů, jejichž cílem je zjistit, zda je klient věrohodný nebo ne. Ve chvíli, kdy služba ověří věrohodnost klienta, se služba pomocí anonymizovaného okruhu připojí k rendezvous bodu a odešle tajný jednorázový řetězec. [20]

V tuto chvíli rendezvous point ověří oba jednorázové řetězce (jeden od služby a jeden od klienta). Pokud se oba řetězce shodují, rendezvous point spojí klienta a službu, kteří si mohou následně předávat end-to-end šifrované zprávy. [20]

Celé spojení probíhá mezi šesti relay. Tři relay vybral klient s tím, že třetí je rendezvous point a tři relay využila služba pro připojení na rendezvous point. [20]

Kapitola 4

Nejznáměji algoritmy hejnové inteligence

V části State of Art jsem nastínil problematiku C&C serveru. V této práci se pokusím C&C server nahradit hejnovou inteligencí a to tak, že každý klient bude zároveň sloužit jako server a všichni klienti budou na stejné úrovni.

Hejnová inteligence spadá pod oblast umělé inteligence, přesněji pod bioinspirované algoritmy.

Algoritmy hejnové inteligence se inspirují chováním hmyzu a zvířat. Důležitou vlastností hejnové inteligence je to, že samotnou úlohu není schopen vyřešit jedinec sám, ale je k tomu potřeba kooperace celého hejna. Hejnovou inteligenci můžeme také znát pod pojmem kolektivní inteligence.

Kolektivní inteligence lze popsat následujícími body:

1. hejno se chová jako celek, aniž by mělo jednu centrální jednotku
2. jedinci si neustále vyměňují informace mezi sebou
3. jedinci reagují samostatně na základě informací od svých sousedů

Hejnové algoritmy se nejčastější využívají pro určitou optimalizační úlohu. Jedince označujeme jako agenty, v přírodě to může být např. mravenec, včela, pták a další. Tyto algoritmy lze označit jako iterativní. [21]

Abychom algoritmus mohli označit hejnovou inteligenci, musí splňovat následujících 5 podmínek:

1. povědomí o okolí – agent by měl reagovat s okolím
2. autonomnost – každý agent by měl být schopen se chovat autonomně
3. solidarita – když je nějaký úkol splněn, agenti by měli být schopni autonomně řešit jiné úkoly
4. rozšiřitelnost – systém, který označíme jako hejnová inteligence, by měl být schopný prozkoumávat další části prostoru možných řešení
5. robustnost – odstranění jedince by nemělo ohrozit chod systému, v lepším případě by měl být systém schopen jedince obnovit

4.1 Ant Colony Optimization (ACO)

Tento algoritmus byl představen roku 1992 Marcem Dorigem v jeho disertační práci. [22]

Algoritmus je inspirován mravenčími koloniemi, které se snaží najít zdroje potravy. Jinými slovy napodobuje chování mravenců v přírodě, kteří se snaží najít nejkratší cestu k potravě. Jedná se o hledání nejkratší cesty v grafu.

Na začátku je pohyb mravenců náhodný. Pokud mravenec nalezne potravu, začne uvolňovat po cestě směrem zpět do mraveniště feromony. Tyto feromony potom přitahují další mravence. Čím více mravenců putuje po označené cestě, tím bude feromonová stopa vetší. Feromony se na cestách postupně vypařují. Na kratších cestách feromony vydrží déle. Díky tomu postupně zanikají delší cesty, které nejsou pro mravence dostatečně atraktivní.

4.2 Particle Swarm Optimization (PSO)

PSO je výpočetní metoda, která napodobuje hejno ptáků, popřípadě ryb, za účelem optimalizace nějaké úlohy. Algoritmus představili James Kennedy a Russell Eberhart v roce 1995. Dodnes se jedná o jeden z nejčastěji využívaných optimalizačních algoritmů. [23]

Algoritmus PSO, tak jako i jiné evoluční algoritmy, využívá označení populace a jedinec v této populaci je pak označován jako částice nebo agent. Na začátku algoritmu máme náhodně nainicializované agenty.

Hejno složené z agentů létá nad nějakou krajinou, což reprezentuje prostor možných řešení a hledá nejvyšší vrchol, tedy např. kopec, popřípadě strom.

Částice, neboli agenti, mají svoji pozici zadanou pomocí souřadnic a rychlost. Jedinec si zároveň pamatuje svou nejlepší pozici a zná hodnotu své účelové funkce.

Každý jedinec analyzuje prostor možných řešení. Ve chvíli, kdy nějaký jedinec objeví nejvyšší vrchol, uloží hodnotu do přístupné paměti celého hejna označované jako gBest a zbytek hejna ho na toto místo následuje.

Zároveň se každý jedinec snaží najít lepší řešení, než je jeho předchozí řešení. Tato hodnota je potom uložena jako pBest.

Algoritmus můžeme dále přizpůsobit na základě toho, jak jedinec bude dále pokračovat.

- Individuální – bude pokračovat po své vlastní cestě
- Konzervativní – bude následovat gBest
- Flexibilní – bude následovat jedince, který měl nejlepší řešení

4.3 Glowworm Swarm Optimization (GSO)

Optimalizaci pomocí roje světlušek je dalším algoritmem, který využívá kolektivní inteligenci. Algoritmus byl poprvé publikován v roce 2016. Technika slouží k hledání globálního optima účelové funkce. Opět využíváme agenty, kteří létají v prostoru a na začátku jsou náhodně rozmístěni. Agenti se vyznačují svojí svítivostí, kterou udává hladina luciferinu. Hladina luciferinu je pak přímo výstupem účelové funkce. Svítivost světlušek láká kořist, popřípadě další světlušky. Agenti nemají žádnou paměť, takže neudržují žádné informace, tak jako v případě PSO. [24]

4.4 Artificial Bee Colony Algorithm (ABC)

Optimalizace včelím rojem je dalším z algoritmů inspirovaným hejnovou inteligencí ze skutečného světa. V tomto případě se zaměřuje na chování včel. Tento algoritmus představil Dervis Karaboga v roce 2005. Algoritmus se inspiruje způsobem komunikace včel pomocí tance, který využívají při hledání potravy.

Algoritmus slouží pro řešení multidimenzionálních a multimodálních optimalizačních problémů.

V algoritmu jsou definované zdroje potravy, které se hodnotí na základě množství nektaru. Včely jsou rozděleny na tři typy: dělnice, vyčkávající a průzkumníci. Dělnice se označují jako zaměstnané včely, které jsou spojené s daným zdrojem potravy, jež v danou chvíli objevují. Nezaměstnané včely neustále kontrolují zdroje potravy, které by mohly využít. Nezaměstnané včely se dělí na průzkumnice a vyčkávající. Průzkumnice prohledávají okolí hnízda a hledají nové zdroje. Vyčkávající včely čekají v hnízdě a získávají informace od zaměstnaných dělnic. Komunikace včel probíhá na základě tance na taneční plošině. Tento tanec provádí dělnice. Čím je tanec delší, tím je zdroj výnosnější. Přihlížející si následně vyberou nejlepší možný zdroj potravy, který budou využívat. Vyčkávající si mezi sebou předávají informace na taneční plošině. Průzkumníci náhodně prohledávají prostor a objevují nové zdroje potravy. Kvalita daného řešení se hodnotí na základě množství nektaru. Pokud se po několika iteracích množství nektaru na dané pozici nezvětší, dělnice je vyslána jako průzkumník, vybere si náhodné místo v prostoru a stane se opět dělnicí. [25]

4.5 SOMA (Self-Organizing Miragrating Algorithm)

Soma byla poprvé představena v roce 1999 profesorem Ivanem Zelinkou. Je založena na spolupráci a soutěžení jedinců, kteří řeší nějaký úkol. Dochází ke střídání dvou fází: spolupráce a soustředění. Jedinci prohledávají prostor možných řešení a neustále hledají lepší řešení. Zde si můžeme všimnout typickou vlastnost algoritmů využívající hejnovou inteligenci. Kromě toho má však mnoho jiných odlišností. Jedinci zároveň tzv. migrují přes prostor možných řešení. Somu můžeme ovlivnit pomocí řídicích parametrů, které dělíme na řídicí a ukončující. Řídicí ovlivňují kvalitu běhu algoritmu, tedy jak rychle běží. Ukončovací pak slouží pro ukončení běhu algoritmu, tedy např. kolik má proběhnout

migračních kol. Smysl spočívá v tom, že částice tvořící nějakou skupinu následují vedoucího jedince. Zde si lze všimnou podobnosti s PSO.

4.6 Inspirace

Hejnová inteligence je jev, který lze pozorovat u velkého množství agentů, kteří vzájemně spolupracují na řešení nějakého úkolů. V současných počítačových vědách se pod pojmem hejnová inteligence chápe množina algoritmů, která využívá těchto principů za účelem optimalizace složitých problémů. Nutno říct, že takovýto pohled na hejnovou inteligenci je značně specifický a samotný fenomén hejnové inteligence nemusí být nutně chápán jako optimalizace. Výrazným rysem hejnové inteligence je komunikace a sdělování informací mezi agenty. V rámci této práce je vytvořen hybridní malware, který se na jednu stranu tváří jako klasický botnet s CNC serverem, ale na druhou stranu má v sobě i prvky hejnové inteligence ve smyslu komunikace a výměny informací mezi agenty. Tato komunikace a výměna informací slouží hlavně k tomu, aby se příslušný payload dostal k agentovi, který ho má vykonat. Lze je tedy i řetězit. Cílem tedy nebylo dělat hluboké teoretické výzkumy v oblasti hejnové inteligence, ale vzít některé její základní hlavní rysy a ty aplikovat na experimentálním malware, vytvořeném v této práci.

Kapitola 5

Python

O Pythonu se většinou mluví jako o interpretovaném jazyku. Nicméně na jeho pozadí dochází ke kompilaci do takzvaného bajtkódu. Tento bajtkódu je následně předán Python Virtual Machine, který ho interpretuje. Python je tedy spíše kompilovaný-interpretovaný jazyk.

Pokud se bavíme o Pythonu většinou se jedná o CPythonu, což je takzvaná referenční implementace a je napsaná pomocí jazyka C a Pythonu. Je to implementace, ze které následně programátoři mohou vytvářet vlastní implementace jazyka Python. Z CPython se nejdříve kompiluje do bajtkódu, který se následně vykoná. Tuto implementaci vytvořil Guido van Rossum.

5.1 Různé způsoby spuštění Python kódů

5.1.1 Cython

Cython (pozor nejedná se o výše zmíněný CPython) je samostatný programovací jazyk, který rozšiřuje klasický Python.

Cython překládá kód z Pythonu do C. Python kód lze překládat buď do C nebo do nativního kódu. Cython přidává statické typování, což může některé části kódu výrazně urychlit. Jeho využití spočívá hlavně v optimalizaci některých kritických částí kódu, pro které se vytvoří kód v C namísto kódu v Pythonu, tedy v Cythonu vytvoříme modul, který následně voláme z interpretovaného Pythonu. [26]

5.1.2 Jython

Jython je implementace Python, u které místo Python Virtual Machine využívá Java virtual machine. Díky tomu mohou vývojáři využívat libovolnou třídu a rozhraní Javy v jejich Jython kódu. Jython kód lze spustit pouze na počítačích s nainstalovaným JVM. Jython stále využívá Python 2. [27]

5.1.3 IronPython

IronPython je implementace Pythonu, která pro svůj běh využívá .NET Framework. Python vývojáři tak mohou využívat nepřeberné množství knihoven .NET Frameworku. Tuto implementaci původně vyvíjel samotný Microsoft, nicméně v roce 2010 od vývoje upustil. Dnes se o projekt starají dobrovolníci na webu Github. Projekt podporuje Python 2. V době psaní této práce byla k dispozici alpha verze IronPythonu pro Python 3.4. [28] [29]

5.1.4 PyPy

Jedná se o just-in-time compiler, jehož výsledný kód může být až několikanásobně rychlejší než referenční CPython. Jedná se o náhradu za CPython. [30]

5.1.5 Brython

Je implementace Pythonu běžící v prohlížeči. Brython naimportujeme jako knihovnu v JavaScriptu na webové stránce. Knihovna se na pozadí postará o překlad Pythonu do JavaScriptu, který potom stránka potom vykoná. [31]

5.1.6 Nuitka

Nuitka se označuje jako source to source compiler. Pomocí nástroje Nuitka můžeme Python kód překládat do C/C++ spustitelných souborů nebo zdrojových kódů. Pomocí tohoto nástroje můžeme také vytvářet samostatně spustitelné aplikace, které poběží na počítačích, které nemají nainstalovaný Python. Nástroj je kompatibilní s velkým množstvím verzí Pythonu. [32]

5.2 Převod Python skriptů do EXE

5.2.1 PyInstaller

PyInstaller je pravděpodobně nejznámějším nástrojem pro převod Python skriptů do jednoho EXE souboru. PyInstaller na pozadí analyzuje předaný skript v Pythonu, vyhledá všechny závislosti a nakonec vytvoří buď jednu složku, ve které jsou veškeré soubory dané aplikace nebo jeden soubor. PyInstaller neprovádí kompilaci kódu, ale zabalí všechny soubory včetně samotného Python interpreteru. Jelikož se samotný interpret přibaluje do spustitelného souboru, dojde ke značnému navýšení jeho velikosti na disku. Pro jednoduchý skript typu Hello World má výsledný soubor okolo 7 MB. Pomocí PyInstaller můžeme vytvořit spustitelný soubory pro Windows, Linux a Mac. [33] [34]

PyInstaller podporuje dvě varianty výstupního souboru:

- onedir - výstupem skriptu je složka se všemi soubory a jedním EXE souborem

- onefile - výstupem je jeden EXE soubor. Tento EXE soubor můžeme distribuovat na další počítače. Po spuštění spustitelné aplikace však dojde k extrakci některých souborů do adresáře Temp

Použití: Po vytvoření virtuálního prostřední pomocí nástroje venv (python -m venv venv) a instalaci všech potřebných Python balíčků pomocí nástroje pip nakonec nainstalujeme i pyinstaller. Nástroj lze také naistalovat pomocí pip (pip install pyinstaller), nebo si ho lze ručně sestavit přímo ze zdrojových kódu.

Při tvorbě této práce jsem také zjistil, že některé antiviry, jako např. MS Defender, označují programy, které využívají PyInstaller, jako trojský kůň. PyInstaller je legitimní balíček, takže se jedná o false positive. Na základě informací ze stránky Stack Overflow jsem se dozvěděl, že tento problém nastává, pokud se knihovna PyInstaller instaluje pomocí pip. [35]

Malware vytvořený v jazyce Python, který využíval PyInstaller, ve většině případů používal balíček nainstalovaný pomocí pip. Tvůrci antivirů proto tento balíček označili za škodlivý.

Jednou z možností, jak tento problém vyřešit, je manuální sestavení balíčku PyInstaller pomocí kódů z Githubu.

Použití PyInstalleru je velice jednoduché, stačí využít příkaz: pyinstaller --onefile nazev_skriptu.py

Parametr --onefile vezme veškeré soubory spojené s Python projektem a udělá z nich jeden EXE soubor. Po spuštění souboru dojde k extrakci některých souborů do adresáře AppData. V případě paramametru --onedir se přímo vytvoří složka, která by se v případě --onefile odextrahovala při spuštění aplikace.

5.2.2 Py2Exe

Py2Exe funguje na podobném principu jako PyInstaller. Do výstupního souboru přibaluje kompletní interpret jazyka Python. Před vytvořením spustitelného suboru nejdříve vytvoříme soubor setup.py. Py2Exe bylo využito pro distribuci dřívějších verzí programu BitTorrent (dnešní μTorrent). [36]

5.2.3 Nuitka

Dalším nástrojem, který by se pro tvorbu spustitelného souboru dal použít, je nástroj Nuitka. Narozdíl od dvou předchozích nevytváří EXE soubor s interpretem Pythonu, ale překompiluje skript v Pythonu do C, a k němu nalinkuje knihovnu v Pythonu. Tento proces je komplikovanější, ale může výsledný kód zrychlit a ztížit následnou reverzní analýzu kódu.

V době psaní tohoto projektu byly však možnosti tvorby samostatného spustitelného souboru velmi omezené.

Výsledný main.exe je nutné distribuovat s knihovnou python39.dll. V následujících měsících by však tento problém měl být vyřešen s použitím parametru --onefile. V době psaní této práce se mi však zkompilování do jednoho výstupního souboru nedařilo.

5.3 Typické funkce Python malwaru

Výhoda Pythonu spočítá ve velkém množství knihoven třetích stran a ve vykonávání kódu za chodu. V podstatě pro libovolnou funkcionalitu existuje nějaký balíček.

5.3.1 Snímek obrazovky

Pro tvorbu malwaru se může například hodit možnost pořizovat snímek obrazovky. Balíček mss umožňuje vytvořit screenshot na platformě Windows, Linux i Mac.

Tvorba screenshotu je závislá na platformě. Pokud bych se chtěl zbavit závislosti na tomto balíčku, musel bych v případě kódu pro Windows využít možností Win32 API. [37] [38]

5.3.2 HTTP Request

Většina dnešní malwarů využívá nějakým způsobem HTTP requesty. Requesty se mohou využít například pro stažení nové verze payloadu popřípadě pro komunikaci se vzdáleným C2 serverem. Pro tvorbu HTTP požadavků v Pythonu lze využít balíček Requests. [39] [38]

5.3.3 Přístup k Win32 API

V Pythonu lze přistupovat také k Windows API. Win32 API můžeme využít například pro odchytávání kláves pro možnosti keyloggeru, pro zápis a čtení z Windows registrů nebo pro již výše zmíněný snímek obrazovky. Pro práci s Win32 API je k dispozici balíček pywin32. [40]

Na internetu je k dispozici koncept keyloggeru napsaného v Pythonu. Pokud bychom vytvářeli nějaký komplexnější malware a do počítače bychom i tak zavedli interpret Pythonu, může to být velmi užitečné, nicméně pokud by měl v počítači být keylogger sám o sobe, zvolil bych pravděpodobně keylogger napsaný např. v C#. [41]

5.3.4 Šifrování

Další důležitou funkcionalitou je šifrování. Pro šifrování můžeme využít knihovnu pycryptodome, která vychází z již dále nepodporované PyCrypto. Pycryptodome lze využít pro symetrické šifrování tedy například AES, ale i asymetrické šifrování jako je RSA. [42] [43]

5.4 Spouštění kódu za chodu

V Pythonu lze vykonávat kód za chodu. Python interpretu předáme řetězec s kódem a interpret pak kód přímo vykoná. Dynamický kód lze spouštět různými způsoby.

5.4.1 Exec

Tato funkce slouží pro dynamické spuštění příkazu. Předaný kód může být buď objekt nebo string. Pokud se jedná o objekt, bude tento kód přímo vykonán. V případě řetězce dojde ještě ke kontrole syntaxe kódu. U funkce exec se bavíme o příkazu, tedy o nějaké činnosti, která má být provedena. Může se jednat o jeden příkaz nebo celý blok příkazů. Funkce exec ignoruje návratovou hodnotu funkce a pokaždé vrací None. [44]

5.4.2 Eval

Funkce eval slouží k dynamickému vykonání výrazu. Hodí se v případě, kdy potřebujeme na vzdáleném počítači vykonat nějaký kód a zároveň vyžadujeme jeho výstup. Funkce eval tedy vyhodnotí nějaký výraz a vrátí jeho výstupní hodnotu. [45] [38]

5.4.3 Subprocess

Python kód můžeme také spustit pomocí nového procesu, což může být alternativa ke dvěma výše zmíněným funkcím. V tomto případě se nemusí jednat čistě o kód v Pythonu, ale můžeme využít libovolný programovací jazyk, např. C#, popřípadě v PowerShell. Subprocessu můžeme předávat parametry a získávat jeho výstup. Tato funkcionalita umožňuje tvůrcům malwaru měnit chování malwaru za chodu. [46]

5.4.4 Shrnutí

V produkčním prostředí mohou funkce jako eval či exec napáchat velké škody, nicméně pro tvůrce malwaru jsou tyto funkce jako stvořené. V kompilovaných malwarerech se častokrát vyskytuje modul, který umožňuje spouštět skripty přímo za chodu. Jedním z příkladů může být malware Flame, který byl napsán v C, ale využíval interpret Lua pro spouštění Lua skriptů. [38]

5.5 Fileless malware

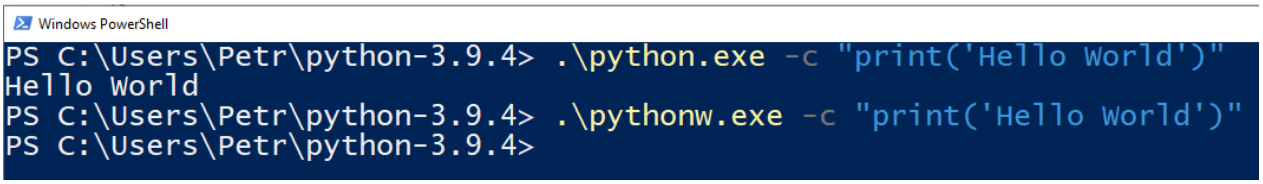
Interpret Pythonu, tak jako PowerShell, umožňuje vykonávat kód přímo z příkazového řádku, aniž bychom k tomu potřebovali soubor.

Kód v Pythonu předáme interpretu jako řetězec pomocí parametru -c

Fileless malware demonstruje obrázek níže. V ukázce jsem využil embed instalaci Pythonu, kterou jsem získal z oficiálních stránek. Jedná se o portable instalaci, takže ji není nutné instalovat.

Archiv v sobě obsahuje dvě verze interpretu Pythonu:

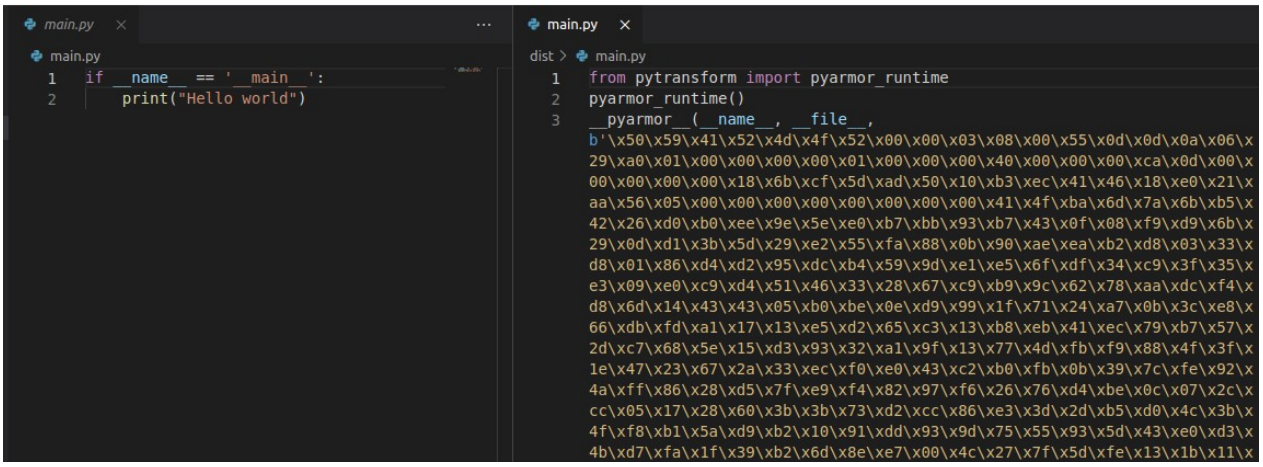
- python.exe - informuje uživatele o všech výjimkách a chybách
- pythonw.exe - výstup potlačuje a taktéž skryje konzoli, což je pro tvůrce malwarů vyhovující



Obrázek 5.1: Spuštění Python kódu přímo z příkazového rádku

5.6 Obfuskace kódu

U většiny dnešních malwarů autoři využívají možnosti obfuskace pro ztížení čitelnosti kódu. Tyto možnosti existují i v Pythonu. Jedním z možných nástrojů je PyArmor. PyArmor vezme skript v Pythonu (příkaz: pyarmor.exe obfuscate main.py) a vygeneruje obfuskovaný kód. Obfuskovaný "Hello World" program je vidět na obrázku níže. [47]



Obrázek 5.2: Obfuskovaný skript pomocí PyArmor

Výhoda PyArmor spočívá v tom, že dokáže spolupracovat s výše zmíněným PyInstallerem pomocí kódu: pyarmor pack main.py

PyArmor je však shareware a při trial verzi povolí obfuskovat jen soubory do velikosti 32768 bajtů. Toto omezení je dostačující, pokud uživatel obfuskuje pouze skripty v Pythonu. Pokud se však pokusí o obfuskaci ve spojení s PyInstallerem, bude potřebovat placenou verzi.

5.7 Ukázky malwaru v jazyce Python

5.7.1 SeaDuke

Skupina malwarů Duke se již vyskytuje několik let. Jedná se o komplexní platformu, která stojí na jádře CozyDuke. Za malwary z rodiny Duke stojí skupina Cozy Bear (APT 29), která se specializuje na špionáž především vládních organizací. Další malware s příponou Duke je pak další komponenta. Velmi zajímavý byl příklad okolo OnionDuke, což byl malware provozovaný na Tor Exit Nodu, který měnil binární soubory, které přes daný uzel prošly. Pokud si tedy uživatel sítě Tor stáhl spustitelný soubor a exit node byl právě tento nakažený uzel, stáhl si do počítače binární soubor nakažený předtím neznámým malwarem. [48]

SeaDuke je však první malware z této rodiny, který využíval jazyk Python a byl navržen, aby fungoval na platformě Windows a Linux.

Seaduke je trojský kůň, který využívá výše zmíněný PyInstaller spolu s UPX packerem. Malware využíval několik úrovní šifrování a obfuskace a obsahoval několik metod persistence na Windows. Pracoval v rámci C2C.

SeaDuke operoval v letech 2015–2016 a byl využit např. pro útok na Democratic National Committee (DNC). Jednalo se o velmi sofistikovaný útok. DNC bylo napadeno dvěma skupinami, výše zmíněnou APT29 a APT 28. [49]

5.7.2 PWOBOT

PWOBOT je podobný malwaru Seaduke. Využíval PyInstaller pro spuštění na Windows. Operoval v letech 2013–2015 a cílil na evropské organizace, nejčastěji v Polsku. Do počítače se dostal při stažení nakaženého souboru z webové služby pro sdílení souborů.

Obsahoval různé metody persistence na Windows. Pro automatický start malwaru při startu systému využíval klíč: HKCU/SOFTWARE/Microsoft/Windows/CurrentVersion/Run/PWO[verze_malwaru]

Malware obsahoval několik modulů:

PWOLauncher - stahoval a spouštěl soubory

PWOHTTPD - na počítači oběti otevřel HTTP server, přes který mohl autor malwaru počítač ovládat

PWOKeyLogger - odchytával klávesy

PWOMiner - těžil kryptoměny

PWOPyExec - spouštěl další Python kód

PWOQuery - odesílal dotazy na vzdálené URL a vracel výsledky

Malware využíval Tor k tunelování veškerého přenosu se vzdálenými servery útočníka. [50]

5.7.3 PyLocky

PyLocky je variace na ransomware Locky, který se šířil v roce 2016. Narozdíl od Lockyho je PyLocky napsán v Pythonu. Do počítače se dostal typicky pomocí škodlivé e-mailové přílohy. Opět využívá PyInstaller pro spuštění na Windows jako samostatný EXE soubor. Kromě PyInstaller využívá také Inno Installer pro vytvoření instalačního balíku. Cílil na různé země (USA, Francie, Italie, Korea). Obsahoval anti-sandbox pravidla, pomocí kterých se snažil zkomplikovat práci při analýze malwaru. Komunikoval v rámci C2C a využíval šifry 3DES. [51]

Zdrojové kódy PyLocky chvíli po začátku spamové kampaně unikly na internet.

Skupina Talos Intelligence provedla analýzu malwaru a vytvořila decryptor, který byl schopen obnovit původní soubory uživatelů. [52]

5.7.4 PoetRAT

Trojský kůň, který je aktivní v průběhu pandemie COVID-19. Cílí především na azerbájdžánskou vládu a energie.

Malware sbíral a kradl informace o ICS/SCADA systémech, které ovládaly větrné turbíny. Malware se do počítače dostal pomocí škodlivého Word dokumentu. Malware měl různé možnosti pro získávání dat, včetně získávání dat z FTP, pořizování snímku z webkamery, zachytávání znaků z klávesnice, získávání dat z webových prohlížečů a kradení uživatelských hesel. Autor malware je zatím neznámý. Tým vědců z Talos Intelligence pojmenoval malware PoetRAT na základě odkazů na Williama Shakespeara, které v kódu našli. [53]

Tento malware má oproti výše zmíněným malwarům hned několik rozdílů.

Malware byl přímo připojen k Word dokumentu. U předchozích malwarů se většinou payload stáhl při spuštění Word dokumentu. Zde se však při spuštění škodlivého Word dokumentu celý dokument načte do paměti a z konce souboru se vezme cca 7 MB velká část, která se následně nakopíruje zpět na disk. Jednalo se o ZIP archiv, který obsahoval interpret Pythonu a samotný kód v Python pro Remote Access Tool. Zip byl odextrahován pomocí makra ve Wordu a makro po extrakci ZIPu zároveň spustilo skript Launcher.py. Launcher otestoval, zda se nejedná o sandbox, na základě kontroly, že disk je větší než 62 GB. V případě, že detekoval sandbox, přepsal skripty malwaru souborem License.txt a ukončil se. Kódy v Pythonu byly obfuskovány pomocí nástroje PyMinifier. [54]

Další zajímavostí malwaru je to, že nepoužíval PyInstaller, ale samostatný interpret Pythonu. Díky tomu, že malware vkládá do počítače interpret jako takový, narozdíl od PyInstalleru, mohou interpret Pythonu využívat i další komponenty malwaru a hacking nástroje.

Malware využíval další nástroje jako pypykatz, WinPwnage, Nmap. Kromě dobře známých nástrojů, napsaných většinou v Pythonu, využíval také nástroj dog, který byl napsán v C# a kontroloval aktivitu na pevném disku. Pokud uživatel změnil nějaký soubor na disku, malware kontaktoval svého majitele pomocí emailu nebo FTP. [53]

5.8 Analýza Python malwaru

Kód v Pythonu se překládá do Python bajtkódu (soubory označené .pyc), který je následně vykonáván virtuálním strojem. Taktéž PyInstaller s sebou nenese zdrojové kódy v Pythonu, ale zkompilované kódy v .pyc.

Při analýze .pyc souborů můžeme využít program uncompyle6, který nám vrátí původní zdrojový kód v Pythonu. [55]

Podobně, pokud bych využil py2exe, existuje zde nástroj pro dekompilaci původních zdrojových kódů.

5.9 Závěr kapitoly

V této kapitole jsem shrnul základní předpoklady pro tvorbu malwaru v jazyce Python a představil malwary, které byly pomocí Pythonu vytvořeny.

Malware napsaný v jazyce Python má své kladné i záporné stránky. Mezi ty kladné patří rychlý vývoj, velké množství knihoven třetích stran, možnost psaní fileless malwaru a tvorby multiplatformních škodlivých kódů. Na platformě Windows zatím není dostupný interpret Pythonu v základní instalaci a to je také hlavní problém, se kterým se musí tvůrci malwaru v Pythonu vypořádat.

Pro tento problém existují v podstatě dvě možná řešení. Prvním je využití nástroje PyInstaller, kdy je kód v Pythonu distribuován spolu s interpretem jako jeden soubor. Tuto možnost využívá většina malwarů napsaných v Pythonu. Druhou variantou je nakopírování interpreta přímo do počítače oběti. Výhoda tohoto řešení spočívá v tom, že interpret lze využít pro spuštění dalších nástrojů napsaných v Pythonu, jako např. WinPwnage. O této možnosti jsem se zmínil v části Fileless Malware a využívá ho např. malware PoetRAT.

Další nevýhodou je relativně snadná deobfuskace kódu. Na kteroukoliv z obfuskačních technik aktuálně existuje alespoň částečná deobfuskace. Řešením může být kompilace některých částí do nativního kódu pomocí Cythonu, Nuitky nebo PyPy. Zde však vzniká mnoho problémů spojených s dohledáváním závislostí, což by původně snadný vývoj mohlo velmi zbrzdit.

Kapitola 6

Návrh řešení

6.1 Volba programovacího jazyka

Pro praktickou část této diplomové práce jsem zvolil programovací jazyk Python. Pomocí tohoto jazyka mohou vytvořit jak jednoduchý skript, tak komplexní backendové řešení. Zároveň jsem na začátku vývoje neměl úplnou představu o tom, jak bude finální verze malwaru vypadat a potřeboval jsem jazyk, který bude vhodný pro rychlé prototypování.

Kromě výše zmíněných důvodů mě také zajímalo, jestli takto složité řešení jde vytvořit pomocí jednoho vysokoúrovňového programovacího jazyka, jehož interpret není v základní výbavě Windows.

Pokud bych však vyřešil problém interpretu, vytvořil bych malware, který je ve své podstatě multiplatformní.

6.2 Způsob komunikace přes Tor

Pro komunikaci přes Tor existují v podstatě dvě varianty:

1. komunikace pomocí HTTP
2. torify

6.2.1 Webová aplikace v Pythonu

První a také primární varianta je komunikace pomocí HTTP, a tedy request-response pattern. Pro tento účel byl Tor navržený a jedná se o nejbezpečnější variantu. V tomto případě bych využil komplexní webový framework Django, který je postavený na Pythonu.

Django obsahuje již v základu podporu třívrstvé architektury a objektové relačního mapování. Django se zaměřuje na bezpečnost a škálovatelnost. Dále má framework nachystané komponenty pro přihlašování a také vlastní administraci ve formě Django admin. [56]

V Djangu můžeme také vytvořit API pomocí balíčku Django Rest framework. [57]

Django využiji pro tvorbu webové stránky, která bude sloužit pro vizualizaci nakažených počítačů, tak pro API, které poběží na každém nakaženém počítači spolu se skriptem v Pythonu.

6.2.2 Torify

Druhá varianta je založena na torify. Torify je termín, při kterém se snažíme data nějaké aplikace např. SSH nebo curl přenášet pomocí sítě Tor. Tento přístup je založený na tom, že po spuštění Toru se na počítači otevře proxy, které slouží pro komunikaci přes Tor. Proxy se automaticky otevírá na portu 9050. Právě SSH je pravděpodobně nejčastějším příkladem pro použití torify. Nicméně můžeme využít v podstatě libovolnou službu.

V mém případě bych vytvořil aplikaci v Pythonu a komunikaci bych řešil přes TCP sockety. I v tomto případě bych využil Onion Service, díky kterým bych daný počítač zpřístupnil přes síť Tor.

Spuštění skriptu by potom probíhalo nějak takto: torify python server.py.

Pokud bych využíval nějakou relativně jednoduchou aplikaci, tak by to byla velmi dobrá volba i přesto, že může dojít k unikum některých dat.

6.2.3 Shrnutí

Pro komunikaci klientů využiji první variantu, kdy každý klient bude mít svoje API a svůj program runner.exe.

S tím, že pokud bych potřeboval využít nějaké další spojení s klientem, jako například shell, využil bych právě možnost torify.

6.3 Odesílání dat přes Tor

Velká část dnešního malwaru využívá nějakým způsobem komunikaci přes síť Tor. V této části popíšu způsoby, jak na nakaženém počítači Tor provozovat.

6.3.1 Tor Browser

Toto může být případ tvůrců ransomwaru, kteří chtějí, aby klient zaplatil na nějakou vzdálenou službu. Pro mě se však nejedná o vhodné řešení, protože nechci, aby klient o Toru vůbec něco věděl.

6.3.2 Tor Expert Bundle

Pokud chceme na nakaženém počítači spouštět Tor a zároveň nechceme, aby o tom uživatel věděl. Vystačíme si s balíčkem Tor Expert Bundle. Jedná se o nejmenší možnou instanci Toru, kterou není potřeba nějak instalovat, pouze se spustí pomocí příkazového řádku. Tor Expert Bundle zabírá na disku okolo 7 MB.

Tuto variantu využil například malware KryptoCibule, o kterém jsem se zmiňoval v kapitole State Of Art.

6.3.3 Vlastní implementaci Toru

Na Githubu je možné najít několik projektů, které se zabývají tvorbou vlastní implementace Tor v jazyce Python. Jmenovitě např. TorPy, TorPylle, TinyTor, PyCepa. Nejdále je pravděpodobně TorPy.

TorPy je open-source implementace protokolu Tor v Pythonu.

Knihovna v době psaní této diplomové práce umí pouze komunikaci s Tor Hidden Service 2, nicméně samotný autor na Githubu zmínil, že plánuje implementaci Tor Hidden Service 3. Jak už jsem poznamenal výše, knihovna zatím umí pouze komunikovat se skrytou službou, což je v tuto chvíli největší problém. Moje aplikace však ke svému chodu potřebuje kompletní implementaci Onion Service. Tuto funkcionalitu by TorPy v budoucnu mohl také umět. Autor projektu zmiňuje implementaci Onion Service na hlavní stránce repositáře. [58]

6.3.4 Závěr kapitoly

V době psaní této diplomové práce nebyla k dispozici knihovna v Pythonu, která by uměla komunikovat přes síť Tor a zároveň provozovat Onion Service. Z toho důvodu využiji pro praktickou část diplomové práce balík Tor Expert Bundle.

6.4 Tvorba EXE souboru

Pro vytvoření EXE souboru z jednotlivých Python skriptů jsem zvolil nástroj PyInstaller.

Kromě PyInstaller existují i další nástroje, jako je py2exe a další, které jsem popsal v kapitole 5.2 Převod Python skriptů do EXE na straně 24.

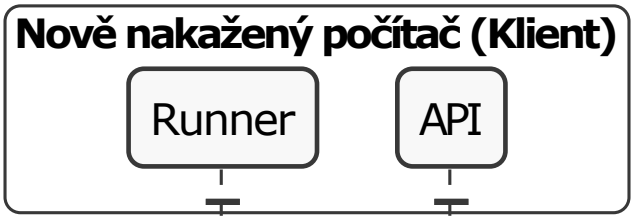
PyInstaller má však nejlepší podporu Django frameworku, který využívám pro lokální API a ControlCenter. Předpokládám, že jeho využití nebude bez komplikací v podobě dohledávání závislostí, nicméně na základě průzkumu se PyInstaller tváří jako nejlepší možná varianta pro moji práci.

Kapitola 7

Implementace

7.1 Klient

Každý nakažený počítač je označován jako klient. Na každém počítači běží 2 oddělené programy: runner.exe a api.exe.



Obrázek 7.1: Nový klient

7.2 Runner

Runner.exe využívá zobecněný databázový model backendové části. Namísto databázových tabulek využívá třídy a zároveň nevyužívá všechny atributy jako například ID, které by mohly ohrozit bezpečnost navrhnutého řešení.

Při spuštění nakaženého souboru se spouští právě runner.exe. Po spuštění se runner postará o nastavení prostředí pro běh malwaru.

7.3 API

Na každém nakaženém počítači běží kromě runneru také API. Pomocí API může nakažený počítač kontaktovat jiné uzly z botnetu a zároveň může přijímat informace od jiných uzlů pomocí svého API. API využívá technologii Django Rest Framework.

API a Control Center sdílí stejný databázový model. Vizualizace databázového modelu je dostupná na straně.

7.4 Control Center

Control Center slouží pouze k ovládání již existujícího botnetu. Při vytvoření Control Center útočník musí u sebe mít spuštěnou API. Ve spuštěné API si následně útočník vytvoří falešného uživatele. Tímto způsobem může získávat informace od všech počítačů botnetu a zároveň zadávat úkoly ostatním členům botnetu. Od tohoto přístupu si slibuji určitou nenápadnost.

7.5 Nákaza

Malware se do počítače dostane stažením a spuštěním programu runner.exe. Po spuštění programu runner.exe dojde k extrakci jeho souborů do složky Temp. Jde o klasické chování programů vytvořených pomocí PyInstalleru. PyInstaller umožňuje do výsledného spustitelného souboru přidat i další soubory, které se také extrahují do adresáře Temp. Z toho důvodu jsem do runner.exe přidal Tor a také prázdnou databázi pro lokální API. Lze přidat i samotné API.

Jedná se o Tor Expert Bundle, což je nejmenší možná forma Toru, která na počítači otevře proxy. Tor Expert Bundle je nezbytný pro spuštění Tor Onion Service. Tor je pouze zabalený v archivu zip. Tor se nemusí instalovat. Pouze se rozbalí na určené místo a spustí se z příkazové řádky.

Runner.exe, tor.exe a soubor s databází se následně překopírují na jiné místo, přesněji do uživatelovy AppData. Na toto místo se následně ještě nakopíruje konfigurační soubor pro Onion službu.

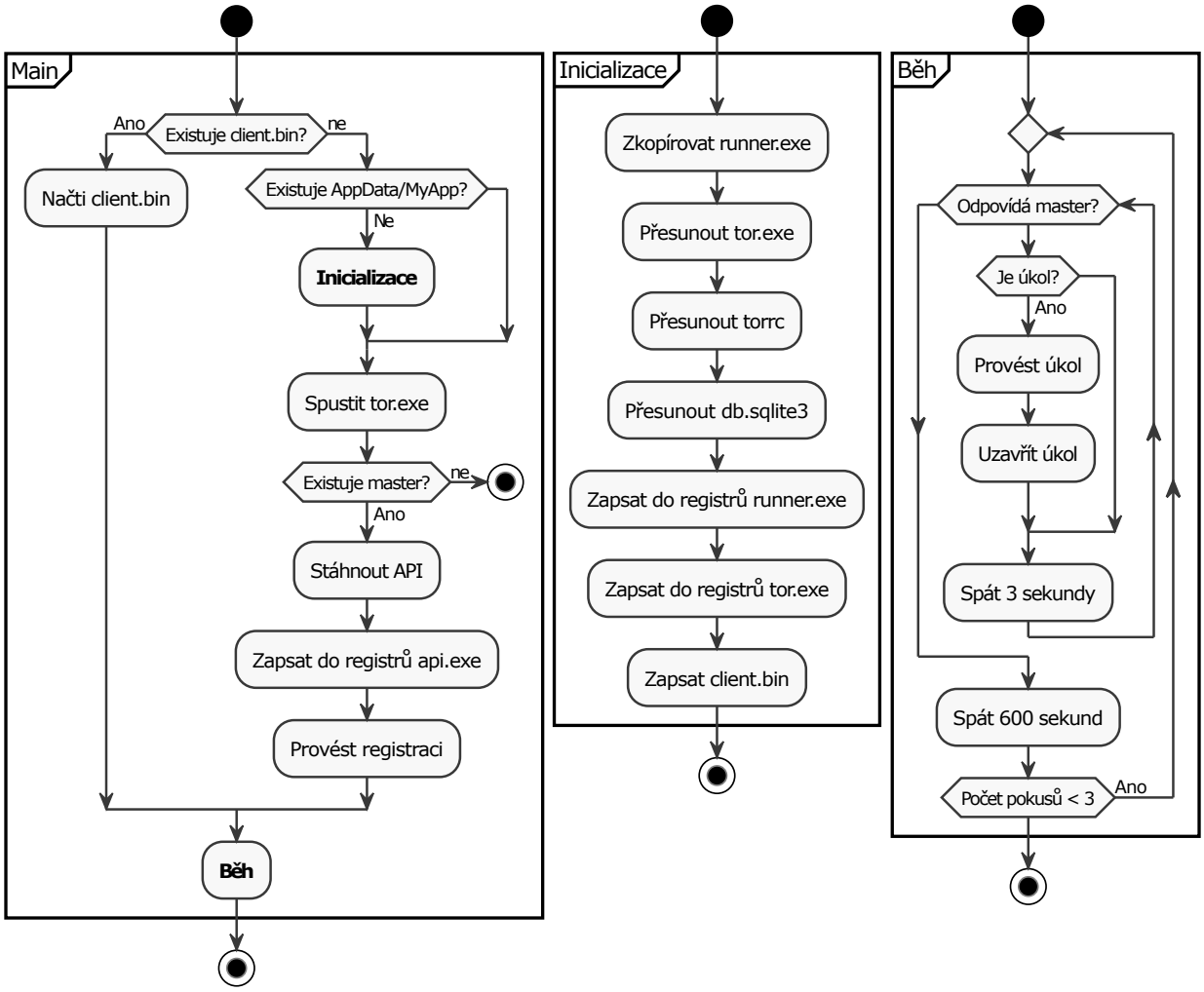
Po nakopírování dat malware zapíše cestu pro runner.exe, tor.exe a api.exe do registrů. Zápis hodnot probíhá do:

\HKEY\CURRENT_USER\SOFTWARE \Microsoft\Windows\CurrentVersion\Run\MyApp

Pro spuštění jednotlivých komponent tedy využívám možností operačního systému, který automaticky spustí všechny tři komponenty při startu počítače.

7.6 Registrace nového klienta

Po kompletním nastavení prostředí pro běh malwaru dojde na registraci nového klienta. Registraci vyvolá runner.exe, který odešle POST požadavek na endpoint /register-me/ své lokální API. Lokální API si nového klienta uloží do své databáze a přepošle informace o novém klientovi master uzlu. Přesněji odešle POST request na jeho API endpoint /new-client/. Master node pak nového klienta distribuuje všem členům botnetu. Následně naplní databázi nového klienta informacemi o všech účastnících botnetu včetně sebe sama. Naplnění databáze probíhá pomocí endpointu /client/. V tuto chvíli dostane API nového klienta odpověď 201, která informuje o úspěšné distribuci informací o novém klientovi. Nakonec ještě lokální API vytvoří nové aktivity typu Establish Connection pro

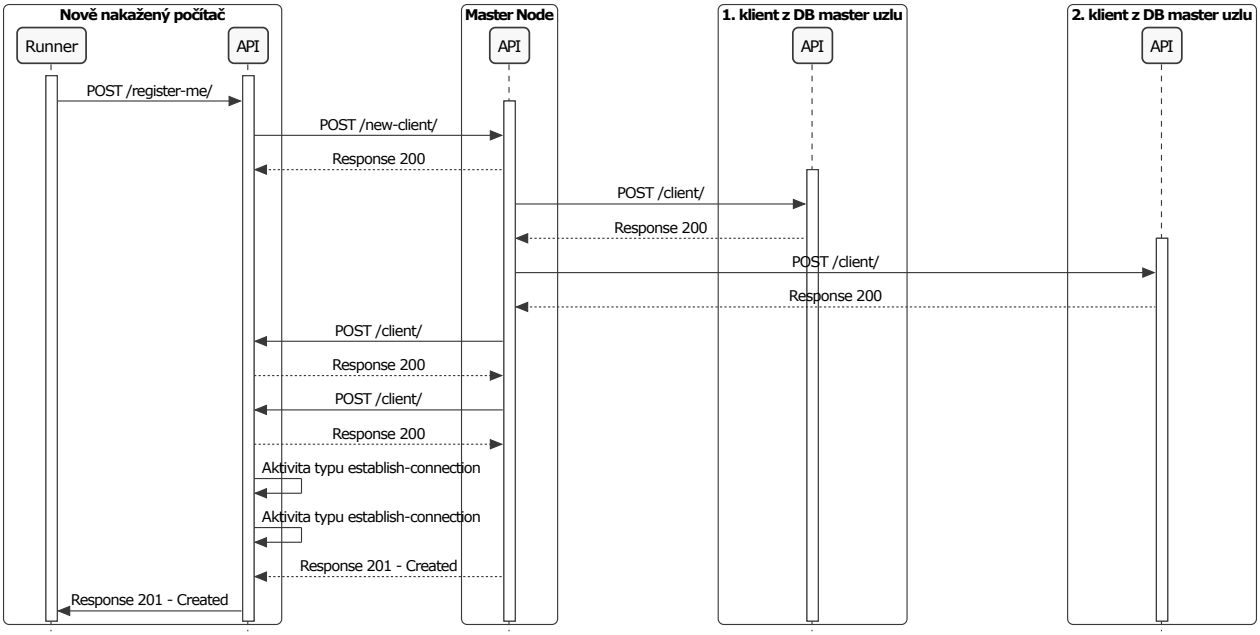


Obrázek 7.2: Runner

každého nového klienta. Až v tomto okamžiku dojde k uzavření celého spojení a lokálně API předá programu runner odpověď 201 - created. Vizualizace registrace nového klienta je vidět na sekvenčním diagramu níže.

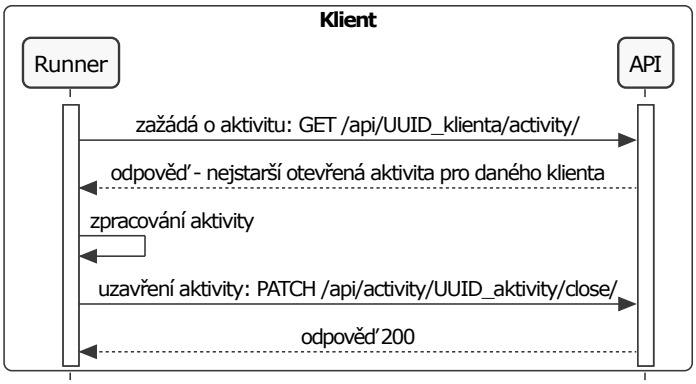
7.7 Získání úkolů

Runner v pravidelných časových intervalech kontroluje, zda nemá na své lokální API zadaný nějaký úkol. Úkoly se jinak označují také jako aktivity. Aktivita obsahuje popis činnosti, kterou má daný klient vykonat. Po vykonání aktivity ji klient uzavře. Pro získání aktivity využívá klient endpoint /api/UUID_klienta/activity/, která vrací nejstarší otevřenou aktivitu. Logika získávání aktivity je tedy čistě na API.



Obrázek 7.3: Registrace nového klienta

Pro uzavření aktivity klient odesílá PATCH request s nově doplněnými informacemi o úkolu, včetně změny hodnoty atributu Open na endpoint /api/activity/UUID_aktivity/close/.



Obrázek 7.4: Získání aktivity

7.8 Vykonání vzdáleného payloadu

V jednom z posledních kroků registrace klienta je vytvoření aktivit typu Establish Connection. Tato aktivita mimo jiné slouží pro získání vzdáleného payloadu. Název Establish Connection je odvozen od toho, že nejprve dojde k navázání spojení se vzdáleným uzlem a až potom klient získá payload, který je uložen v databázi daného vzdáleného uzlu.

Toto vyžádání payloadu probíhá šifrovaně a aby nebyl payload přístupný komukoliv z internetu, není payload dostupný přes požadavek typu GET, ale POST.

Klient, který chce získat payload některého jiného klienta ze sítě, si nejdříve musí ze svojí databáze vytáhnout informace o vzdáleném klientovi včetně jeho veřejného klíče.

Potom vygeneruje nový klíč pomocí symetrické šifry AES. Tento klíč označuji jako klíč relace.

Tento klíč relace se dále zašifruje oponentovi veřejným klíčem. Zašifrovaný klíč pomocí metody POST se odešle na endpoint /establish/ oponentovi API.

Oponent si zprávu přečte, dešifruje pomocí svého RSA privátního klíče a klientovi vrátí payload zašifrovaný pomocí AES klíče.

Jak si čtenář může všimnout, tak oproti HTTPS je toto šifrování zkrácené. Za normálních podmínek by nejdříve měla proběhnout výměna veřejných klíčů a následně až klíče symetrického. Zde se však šifrování využívá spíše jako určitá forma obfuskace.

Myšlenka spočívá v tom, že program runner.exe v sobě bude obsahovat jeden nebo žádný payload a všechny ostatní payloady si bude schopen stáhnout ze vzdáleného zdroje.

Díky tomuto přístupu můžeme vygenerovat několik druhů programu runner.exe pokaždé s jiným payloadem a výrazně tak zvýšit odolnost proti antivirový řešením, protože nebude existovat jen jeden typ malwaru.

7.9 Vybrané metody runnera

Jak už jsem zmínil výše, tak tabulka v databázi je zvizualizovaná pomocí třídy na straně runnera. Kromě tříd Client, Activity, Payload má klient navíc ještě třídy Utility, Encryption a Session.

Třída Utility obsahuje statické metody. Tyto metody nejčastěji využívá třída Client. Nejdůležitější metodou třídy Utility je pravděpodobně make_request(). Tato metoda využívá knihovnu requests pro tvorbu HTTP požadavků. V defaultním stavu je nastavena komunikace pomocí Toru. Vstupní parametr tor udává, jestli se výkonná klasický požadavek, nebo požadavek půjde přes síť Tor. Pro komunikaci přes Tor je potřeba vytvořit objekt requests.Session(). K tomu slouží metoda get_tor_session(). V opačném případě se volá přímo requests a název metody.

Metoda make_request() se využívá například ve třídě Client při registraci runnera u své lokální API, uvnitř metody register_me(), popřípadě při zpracovávání aktivity uvnitř metody process_activity() a nakonec při uzavření aktivity v metodě close_activity().

Třída Encryption obsahuje metody pro práci s RSA a AES. Kromě runnera ji v obdobné podobě využívá také API. Třída se využívá primárně pro šifrování a dešifrování při získávání vzdáleného payloadu. Dá se také využít pro šifrování souborů v případě payloadu.

V této třídě najdeme například metody generate_client_keys(), která slouží pro generování klíčů klienta, rsa_encrypt() pro šifrování a rsa_decrypt() pro dešifrování RSA šifer. Pro AES šifru potom metody aes_encrypt() a aes_decrypt().

Třída Session slouží pro uchování informací o navázaném spojení se vzdáleným klientem.

Třída Payload se využívá jak v případě vykonávání vzdáleného payloadu, tak v případě lokálního payloadu. Instance této třídy se ukládá do binárního souboru payload.bin. Kromě kódu v podobě textového řetězce s sebou nese instance třídy také informace o výstupu a typu payloadu. Třída obsahuje metodu run_payload(), která spouští payload pomocí funkce exec().

7.10 Dostupné API endpointy

```
Request: POST /register-me/
{
    "name" = {String} #uživatelské jméno
    "os" = {String} #operační systém
    "type" = {String} #Guest nebo Owner
    "admin" = {Boolean} #zda má uživatel administrátorská práva
    "created" = {DateTime} #vytvořeno automaticky
    "uuid" = {UUIDv4} #klientem vygenerované UUID
    "public_key" = {String} #veřejný klíč klienta zakódovaný pomocí base64
    "api_url" = {url} #URL adresa klientovi API
    "master_url" = {url} #URL adresa jeho master uzlu
}
Respone: 201 Created
{
    "uuid": {UUIDv4} #zvolené UUID
}
```

Listing 7.1: Vytvoření nového klienta - odesílá klient na své lokální API

```
Request: POST /new-client/
{
    "name" = {String} #uživatelské jméno
    "os" = {String} #operační systém
    "type" = {String} #Guest nebo Owner
    "admin" = {Boolean} #zda má uživatel administrátorská práva
    "created" = {DateTime} #vytvořeno automaticky
    "uuid" = {UUIDv4} #klientem vygenerované UUID
    "public_key" = {String} #veřejný klíč klienta zakódovaný pomocí base64
    "api_url" = {url} #URL adresa klientovi API
    "master_url" = {url} #URL adresa jeho master uzlu
}
```

```
Respone: 200 OK
```

Listing 7.2: Slouží pro rozšíření klienta do botnetu - klientovo API udělá request na endpoint API masteruzlu

```
GET /client/{uuid}
{
    "uuid": uuid_klienta,
}
```

Listing 7.3: Získání klienta na základě UUID

```
Request: POST /client/
{
    "name" = {String} #uživatelské jméno
    "os" = {String} #operační systém
    "type" = {String} #Guest nebo Owner
    "admin" = {Boolean} #zda má uživatel administrátorská práva
    "created" = {DateTime} #vytvořeno automaticky
    "uuid" = {UUIDv4} #klientem vygenerované UUID
    "public_key" = {String} #veřejný klíč klienta zakódovaný pomocí base64
    "api_url" = {url} #URL adresa klientovi API
    "master_url" = {url} #URL adresa jeho master uzlu
}
Respone: 201 Created
```

Listing 7.4: Vytvoření nového klienta

```
GET /activity/{uuid}
{
    "uuid": uuid_aktivity,
}
```

Listing 7.5: Získání aktivity na základě UUID

```
Request: POST /activity/
{
    "name" = {String} #název aktivity
    "type" = {String} #typ aktivity
    "command" = {String} #vstupní parametry pro vykonání aktivity
    "content" = {String} #výstupní parametry provedené aktivity
```

```
    "created" = {DateTime} #vytvořeno automaticky
    "author" = {Integer} #cizí klíč na tabulku Client
    "open" = {Boolean} #signalizuje, zda se jedná o vykonanou aktivitu
    "uuid" = {UUIDv4} #automaticky vygenerované UUID aktivity
    "attachement" = {File} #příloha v podobě souboru, např. pro PrintScreen
}
Response: 201 Created
```

Listing 7.6: Vytvoření nové aktivity

```
Request: PATCH /activity/uuid_aktivity/ #Parametry, které si přeji změnit (stejné
    jako v případě metody POST)
Response: 200 OK
```

Listing 7.7: Změna aktivity

```
Request: POST /payload/
{
    "type" = {String} #typ payloadu
    "code" = {String} #kód v Pythonu
    "uuid" = {UUIDv4} #vygenerované UUID
    "output_type" = {String} #zdali je výstup do souboru
    "output_path" = {String} #pokud je output_type file, zde je uložena cesta k
        souboru
}
Response: 201 Created
```

Listing 7.8: Vytvoření nového Payloadu

```
Request: UPDATE, PATCH /payload/{uuid_payloadu}/
{
    "type" = {String} #typ payloadu
    "code" = {String} #kód v Pythonu
    "uuid" = {UUIDv4} #vygenerované UUID
    "output_type" = {String} #zdali je výstup do souboru
    "output_path" = {String} #pokud je output_type file, zde je uložena cesta k
        souboru
}
Response: 200 OK
```

Listing 7.9: Úprava payloadu na základě UUID

```
Request: GET /status/
Response: 200 OK
```

Listing 7.10: Slouží pro otestování dostupnosti klienta

```
Request: POST /encryption-init/
{
    "path" = {String} #cesta k privátnímu klíči
}

Response: 202 Accepted
{
    "path" = "updated" #cesta k souboru úspěšně aktualizována
}
```

Listing 7.11: Odesílá runner na lokální API po vygenerování šifrovacího páru

```
Request: POST /establish/uuid_klienta/
{
    "msg" = {String} #AES klíč zakódovaný pomocí Base64 a zašifrovaný příjemcovým
        veřejným klíčem
}

Response: 202 Accepted
{
    "ciphertext": {String}, #payload zašifrovaný předaným AES klíčem
    "aesIV": {String}, #inicializační vektor
    "authTag": {String}, #ověření integrtiy
}
```

Listing 7.12: Odesílá nový klient na API vzdáleného klienta

Kapitola 8

Sestavení

Tato kapitola popisuje vytvoření spustitelných souborů pomocí nástroje PyInstaller. Konkretní sestavení spustitelného souboru je následně popsáno v prvním experimentu.

8.1 Runner

Základní vytvoření EXE souboru pomocí nástroj PyInstaller jsem již nastínil v teoretické části. Pro sestavení výsledného souboru je však potřeba doplnit některé parametry. Do spustitelného souboru přibalujeme Tor včetně jeho konfiguračního souboru. Tyto dva soubory jsou zabalené v zipu a jsou přidány PyInstalleru pomocí parametru add-data.

Zároveň je nutné, abychom PyInstaller spouštěli z nadřazené složky a parametry předávali pomocí absolutních cest. Tento krok je nezbytný, aby systém při kopírování souborů obsažených v dočasném adresáři PyInstalleru, nevyhodil výjimku v podobě Permission Denied.

Pomocí příkazu --noconsole skryjeme konzolové okno, aby aplikace běžela na pozadí.

Nakonec ještě přidáme název projektu pomocí parametru name.

```
pyinstaller.exe --onefile --add-data "C:\Users\Petr\Desktop\client_app\tor.zip;."
    --noconsole --name "MyApp" "C:\Users\Petr\Desktop\client_app\main.py"
```

Listing 8.1: Vytvoření EXE pro runner pomocí nástroje PyInstaller

8.2 API

Vytvoření souboru api.exe je o poznání jednodušší, protože do spustitelného souboru není potřeba přibalovat další soubory, tak jak je tomu v případě runneru. U api.exe předáváme PyInstalleru pouze parametr onedir, icon s cestou k ikoně, name s názvem výstupního souboru a cestu k vstupnímu skriptu.

Cestu k vstupnímu skriptu taktéž předáváme pomocí absolutní cesty. V tomto případě vytváříme výstupní soubor ze skriptu server.py.

```
"pyinstaller.exe --onedir --icon="api.ico" --name "api" "C:\Users\Petr\Desktop\api
    \master\server.py" "
```

Listing 8.2: Vytvoření EXE pro API pomocí nástroje PyInstaller

Kapitola 9

Nasazení

Pro nasazení využívám nástroj PyInstaller. PyInstaller využívám jak pro Runner, tak pro API. Na každém počítači budou tedy 2 samostatné programy ve formátu EXE. PyInstaller jsem využil hlavně pro snadné nasazení na Windows. Jak už jsem zmínil v kapitole 5.2 Převod Python skriptů do EXE na straně 24, je tento nástroj dostupný i na jiné platformy.

Pro spuštění Django serveru lze využít zabudovaný vývojový server, který se spouští příkazem manage.py runserver. Osobně jsem tento server zkoušel v prostředí Toru a fungoval. Nicméně samotní vývojáři Djanga doporučují pro produkční nasazení využít jiný server. Jelikož pro klientskou část se zaměřuji na platformu Windows, musel jsem najít odpovídající aplikační server. Nároky na tento aplikační server nebyly nikterak velké, spíše šlo o bezproblémovou funkci v prostředí PyInstalleru.

Na základě průzkumu jsem zjistil, že uživatelé mají největší zkušenosti s aplikačním serverem Cherrypie, který podporuje jak Django, tak Flask. Cherrypie jsem původně chtěl využít také, ale zjistil jsem, že veškeré návody jsou několik let staré a už nefungují. Při dalším průzkumu jsem objevil alternativu v podobě serveru Waitress. Waitress je WSGI server napsaný v Python a ke svému běhu nepotřebuje žádné další balíčky. Lze ho nainstalovat jednoduše pomocí pip a zakomponovat do PyInstalleru. Aplikační server mi zjednoduší nasazení a postará se o vícenásobné připojení klientů v jeden moment. Zde by vývojový server Djanga mohl mít problém.

Kromě instalace knihovny pomocí pip, jsem také lehce upravil soubor manage.py, abych nemusel zadávat číslo portu jako parametr při spuštění aplikace. [59]

Pro nasazení Control Center lze využít i zabudovaný Django vývojový server. Toto je nejjednodušší způsob nasazení. Zabudovaný vývojový server funguje jak pro Windows 10, tak pro Linuxové distribuce a Mac OS X.

Pokud by uživatel chtěl maximální bezpečnost doporučuji využít linuxovou distribuci např. Ubuntu, jako webový server využít NGINX a jako aplikační Gunicorn. Nasazení je však daleko složitější.

Kapitola 10

Konfigurace

10.1 Informace o master uzlu

Komunikace mezi Runner lokální API se řeší čistě na localhostu. Můžeme však nastavit port, ke kterému se Runner připojí a zároveň u API můžeme nastavit port, na kterém bude API naslouchat.

Adresa včetně portu je automaticky nastavena na 127.0.0.1:8000, tato adresa je napevno uložena v programu runner. Zároveň je to výchozí adresa, na které API naslouchá.

Větší problém nastává v podobě adresy master uzlu, která musí být dostupná při spuštění programu runner.exe.

Onion adresu můžeme definovat pomocí vstupních parametrů, popřípadě ji můžeme napevno uložit do runner.exe

Každá z těchto variant má svoje výhody i nevýhody. V případě předávání parametrů máme jednoho univerzálního klienta, ale relativně složité nasazení. Adresu master uzlu je možné předat jako parametr při spuštění. Zde ale nastává otázka, kde by ji runner získal. Spouštění klientů s přidáním vstupních parametrů tedy slouží spíše pro testovací účely, kdy se spouští několik klientů a serverů na jednom počítači.

V případě druhé varianty musíme generovat nové runnery pro další master uzly, ale problémy s nasazením v podstatě odpadají, protože máme jeden spustitelný soubor bez dalších parametrů. Adresa master uzlu je sice napevno zakódovaná v programu, ale tuto adresu můžeme dále obfuskovat, aby bylo její objevení komplikovanější.

Zajímavou alternativou by mohlo být uložení obfuskované URL adresy master uzlu do událostí Windows a využívat tak více než jeden způsob perzistence. Tento způsob je spíše jeden z možných způsobu vylepšení aplikace.

Kapitola 11

Testování a ladění

Spouštění aplikace z prostředí PyInstaller slouží pouze pro nasazení. Pro úpravu kódu je lepší spouštět aplikaci přímo z příkazové řádky pomocí Python interpretu.

11.1 Spuštění Runneru pro testovací účely

Runner spouštíme pomocí main.py. Pokud nepředáme skriptu žádné parametry, bude se runner automaticky připojovat na adresu 127.0.0.1:8000, jako master node se nastaví 127.0.0.1:9779 a domovský adresář pro malware se zvolí MyApp, tedy přesněji C:\Users\Petr\AppData\Roaming\MyApp.

Předání parametrů vypadá následovně

```
python.exe adresa_master_uzlu adresa_klientovi_api adresar
```

Listing 11.1: Spuštění API pomocí Pythonu pro platformu Windows

Název adresáře není potřeba zadávat ve chvíli, kdy máme na počítači jen jednoho klienta a API. V případě více klientů, je vhodné pro nového klienta nastavit vlastní adresář, aby nedošlo k přepsání souborů v adresáři AppData.

```
python main.py 127.0.0.1:9333 127.0.0.1:9779 Storage
```

Listing 11.2: Spuštění API pomocí Pythonu pro platformu Windows

11.2 Spouštění API pro testovací účely

Klasickou Django aplikaci spouštíme pomocí vývojového serveru Django příkazem mange.py runserver.

Pro testování API je však vytvořen samostatný soubor s názvem serve_api.

Jedná se o alternativu k runserveru. Tento soubor jsem využil pro své potřeby. Využívá přímo Waitress, takže minimalizuje případné problémy, které by mohly nastat při převodu pomocí PyInstaller.

```
python master\server.py 127.0.0.1:9779 api Storage
```

Listing 11.3: Spuštění API pomocí Pythonu pro platformu Windows

11.3 Logování

Pro zjištění aktuálního chodu programu je možné využít logování.

V celém programu se místo funkce Print využívá logování pomocí knihovny logging. Logování je v defaultním stavu zapnuté pro výpis do konzole. Pokud je však konzole schovaná, můžeme využít logování do souboru a průběžně si ukládat aktivitu klienta. Toto se může hodit v případě, že vytvoříme nový payload, který bude spuštěn na dálku. Díky logování si můžeme ověřit, že payload byl úspěšně vykonán. Dále se logování může hodit u šifrování, kde může nastat velké množství komplikací především se špatným klíčem. Logování také může posloužit pro lepší pochopení kódu spolu s komentáři. [60]

11.4 Tvorba a použití payloadů

Pro payload existuje na backendu samostatná tabulka Payload, která je na klientovi znázorněna pomocí třídy.

Payload můžeme do runneru vložit přímo jako kód pomocí instance třídy Payload, popřípadě můžeme využít možností PyInstalleru a parametru add-data. V tomto případě předáváme payload jako vytvořenou instanci, která je však zapsaná do souboru. Tímto způsobem můžeme vytvářet runnery, které mají stejnou základní funkcionalitu, ale modulární payload.

Runner po startu odesílá svůj payload na svoje API pomocí endpointu payloads. Díky tomu je škodlivý kód dostupný i pro ostatní klienty botnetu. Škodlivý kód se nevykoná hned, ale musí být spuštěn pomocí aktivity. Aktivity se automaticky vytvoří po registraci klienta do sítě. Aktivity typu establish-connection se vytvoří podle počtu klientů po registraci nového klienta do botnetu. Po vytvoření establish-connection je nakonec vytvořena i aktivita pro vykonání svého vlastního Payloadu. Klient tedy nejdříve vykoná payload všech klientů v sítí a až nakonec vykoná svůj vlastní.

11.5 Dokumentace

Kromě dokumentace v podobě této diplomové práce může čtenář využít dokumentace vygenerované na základě komentářů ve zdrojovém kódu.

V Python existuje velké množství různých nástrojů pro tvorbu dokumentace. Nejznámější je pravděpodobně Sphinx, což je nástroj pro automatické vygenerování dokumentace v Pythonu na základě reStructuredText. Dokumentaci potom můžeme vygenerovat například ve formátu HTML. Tento nástroj jsem také použil pro dokumentaci runneru. [61]

Sphinx funguje dobře pro klasické Python skripty. Pro jednotnost jsem se snažil Sphinx použít také pro Django a Django Rest framework. Dokumentaci se mi povedlo vygenerovat, ale nebyla úplně přehledná.

Pro dokumentaci v Djanga lze využít také Django admindocs, což je modul dostupný přímo v Djangu. Tento modul je založen na Sphinx. Dokumentace je potom dostupná po přihlášení do administrace Djanga v prostředí django-admin. [62]

Možnost zobrazení dokumentace jsem nechal aktivní pouze pro control center. Nicméně v dokumentaci jsou dostupné metody jak pro Django, tak pro Django Rest framework.

Kapitola 12

Experimenty

V této kapitole názorně předvedu výsledné řešení praktické části této práce.

12.1 První experiment: ukázka funkcionality

První experiment demonstruje samotnou funkcionalitu. Od tvorby spustitelných souborů malwaru, přes nákazu a distribuci klienta dalším členům botnetu, až po vzdálené ovládání nakažených počítačů pomocí Control Center.

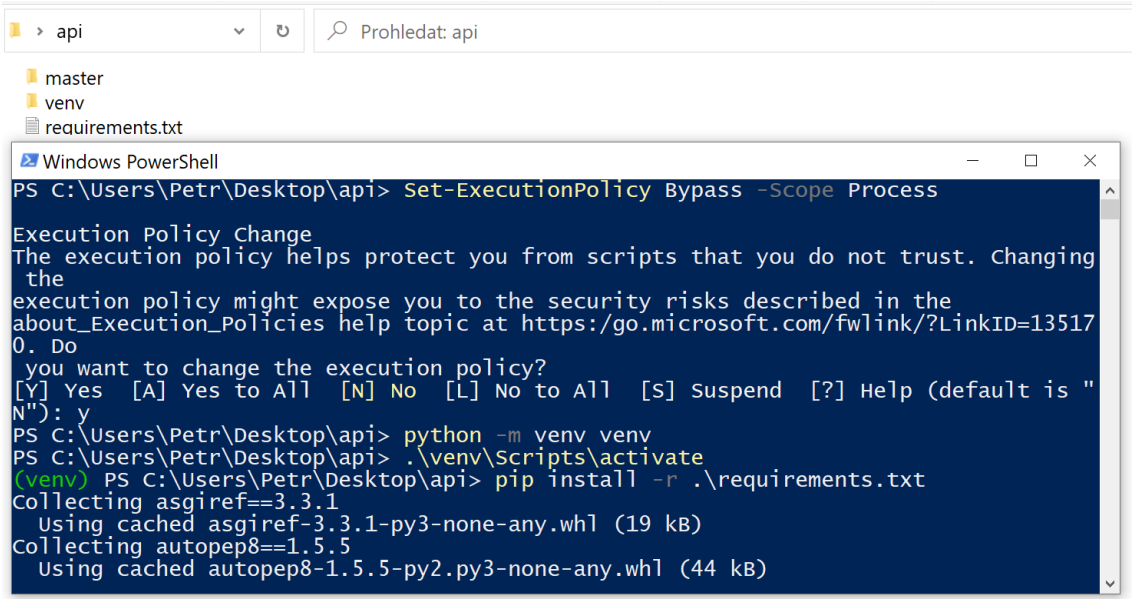
12.1.1 Tvorba klientů

Pro experiment využiji několik virtuálních počítačů. Počítače obětí i počítač útočníka jsou postavené na operačním systému Windows 10. Windows 10 pro počítač útočníka využívám z důvodu zjednodušení tohoto experimentu. V případě skutečného nasazení doporučuji využít technologie zmíněné v kapitole 9 Nasazení, tedy linuxovou distribuci např. Ubuntu, jako webový server využít NginX a jako aplikační potom Gunicorn.

Jako první jsem vytvořil archiv s API. API je totiž minimálně v případě prvního uzlu potřeba předat přímo do runneru.

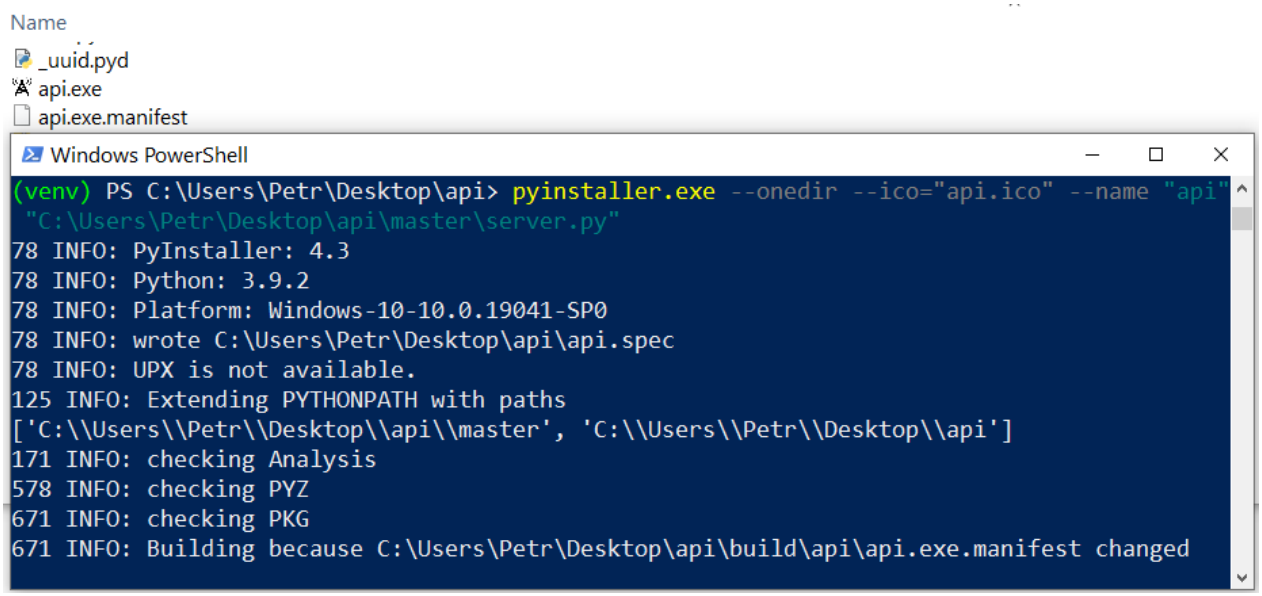
V první řadě bylo potřeba vytvořit virtuální prostředí pomocí příkazu python -m venv venv a nainstalovat všechny balíčky pomocí nástroje pip. Seznam těchto balíčků jsem již předem vyexportoval do souboru requirements.txt. V prostředí PowerShellu bylo ještě potřeba nastavit ExecutionPolicy.

Pro sestavení API jsem v PyInstalleru využil parametr --onedir. Pomocí tohoto parametru se vytvoří jedna složka, ve které jsou všechny soubory projektu, včetně spustitelného souboru. Kromě tohoto parametru jsem předal také cestu k ikoně pomocí parametru --ico a název souboru parametrem --name.



Obrázek 12.1: Vytvoření virtuálního prostředí venv pro API

Tvorba spustitelného souboru pro API je jednodušší než v případě runneru, díky tomu, že veškeré potřebné soubory s sebou nese právě runner.

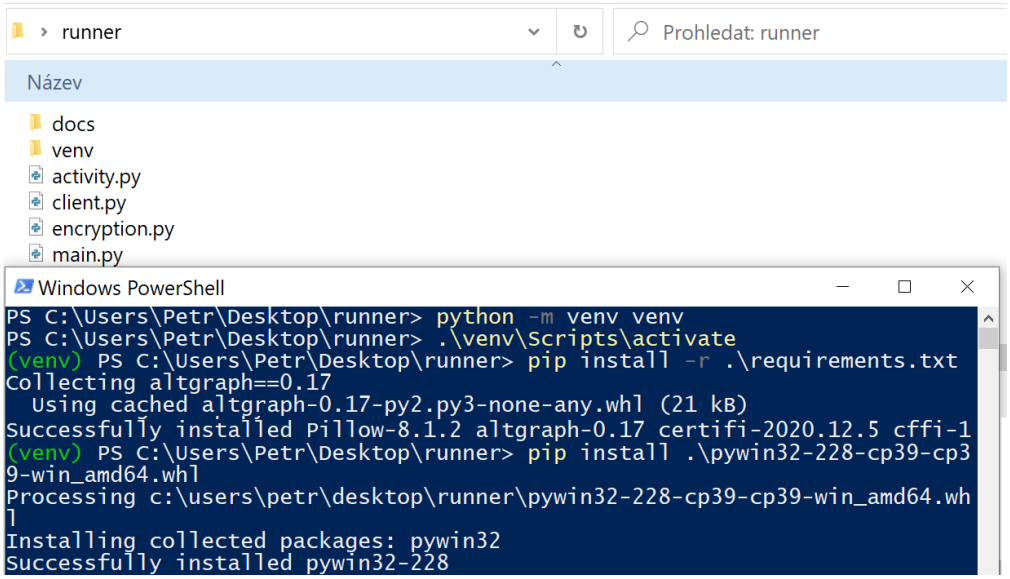


Obrázek 12.2: Vytvoření spustitelného souboru pro API

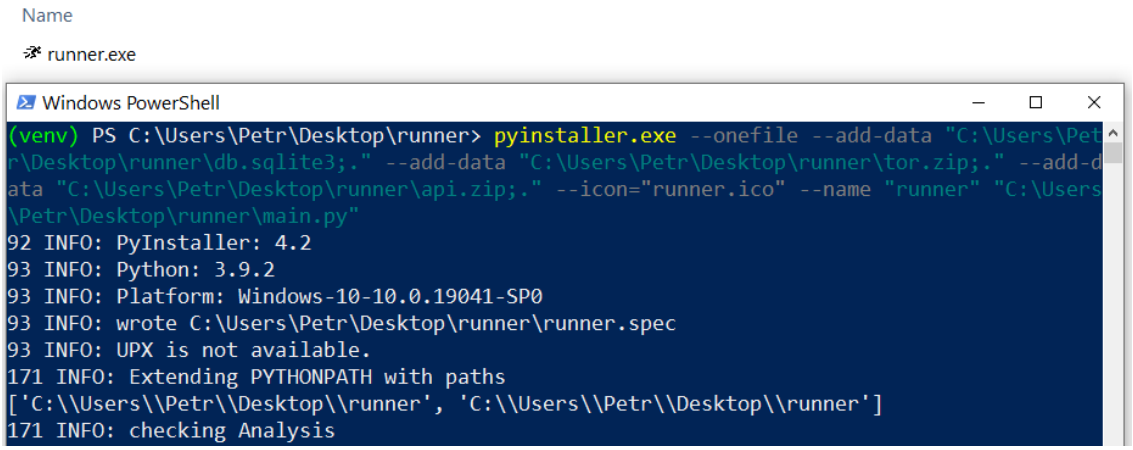
Následně jsem vytvořil runnery. První runner neobsahoval žádný payload, druhý měl jako payload snímek obrazovky.

Opět jsem musel vytvořit virtuální prostředí pomocí příkazu python -m venv venv a nainstalovat všechny balíčky pomocí nástroje pip.

Následně jsem taktéž využil PyInstaller. K příkazu pyinstaller.exe jsem přidal několik vstupních parametrů. Prvním parametrem --onefile jsem nastavil, že bude jeden výstupní spustitelný soubor. Dalším parametrem --add-data jsem přidal do výsledného souboru zmigrovanou databázi pro API s jedním vytvořeným uživatelem s administrátorskými právy. Dalším --add-data jsem přidal zip archiv Toru. Kromě Toru jsem do runneru také přidal api. Posledním --add-data jsem předal cestu k souboru s instancí třídy Payload. Parametrem --ico jsem předal cestu k ikoně spustitelného souboru. Parametrem --name jsem nastavil název výsledného souboru. Jako poslední jsem přidal cestu k Python skriptu, ze kterého jsem vytvořil spustitelný soubor. Runner jsem vytvářel na jiném počítači, čímž demonstruji, že spustitelný kód je přenositelný.



Obrázek 12.3: Vytvoření virtuálního prostředí pro runner



Obrázek 12.4: Vytvoření spustitelného souboru pro runner

12.1.2 Nákaza a distribuce nového uzlu

Vytvořený spustitelný soubor jsem přenesl na počítače obětí a postupně spustil.

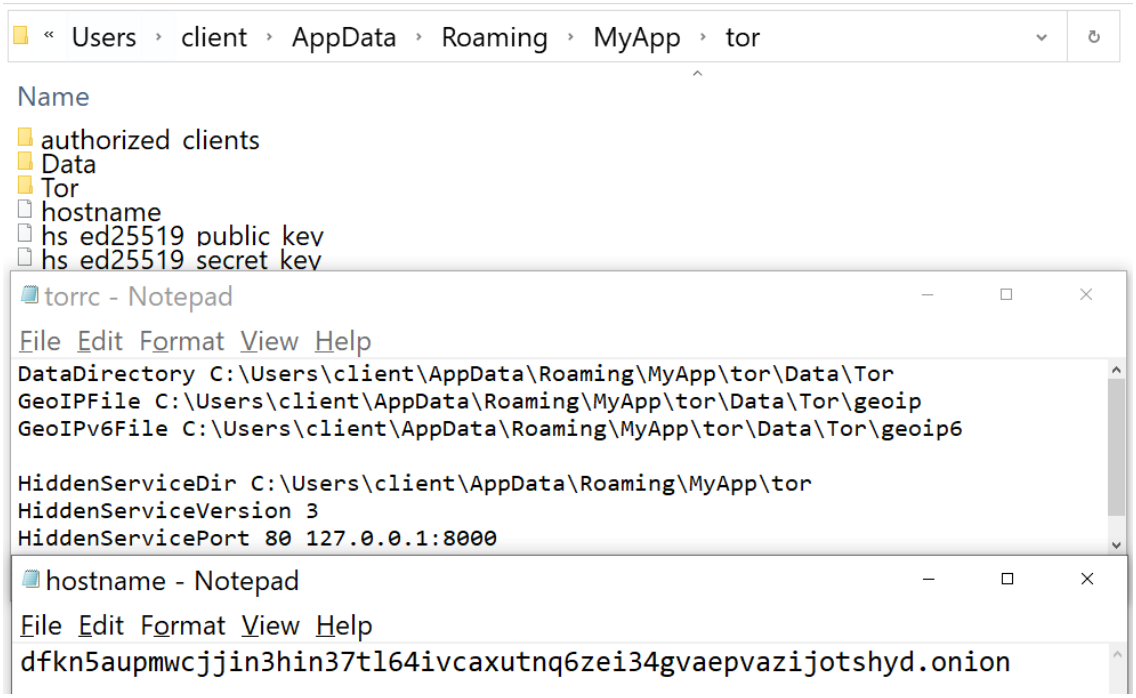
Nákaza probíhala následujícím způsobem. Po spuštění runneru došlo k otestování, zda jsou ve složce AppData k dispozici všechny komponenty malwaru, tedy API, Tor a runner. Pokud kterýkoliv prvek chyběl, došlo k jeho obnovení. Tor a API se obnovily z adresáře, který byl vytvořen po spuštění runneru. Runner se zkopíroval z místa, odkud byl spuštěn.

Taktéž došlo k otestování, zda pro runner a API existují hodnoty v registrech. Pokud chyběly, došlo k jejich obnovení. Každý klient měl ve své složce AppData podsložku media, která sloužila ke zveřejnění souborů ostatním uzlům. API je uloženo právě v této složce. Po jeho přesunutí je tedy API automaticky dostupné dalším klientům pomocí adresy http://adresa_uzlu:port/media/api.exe.

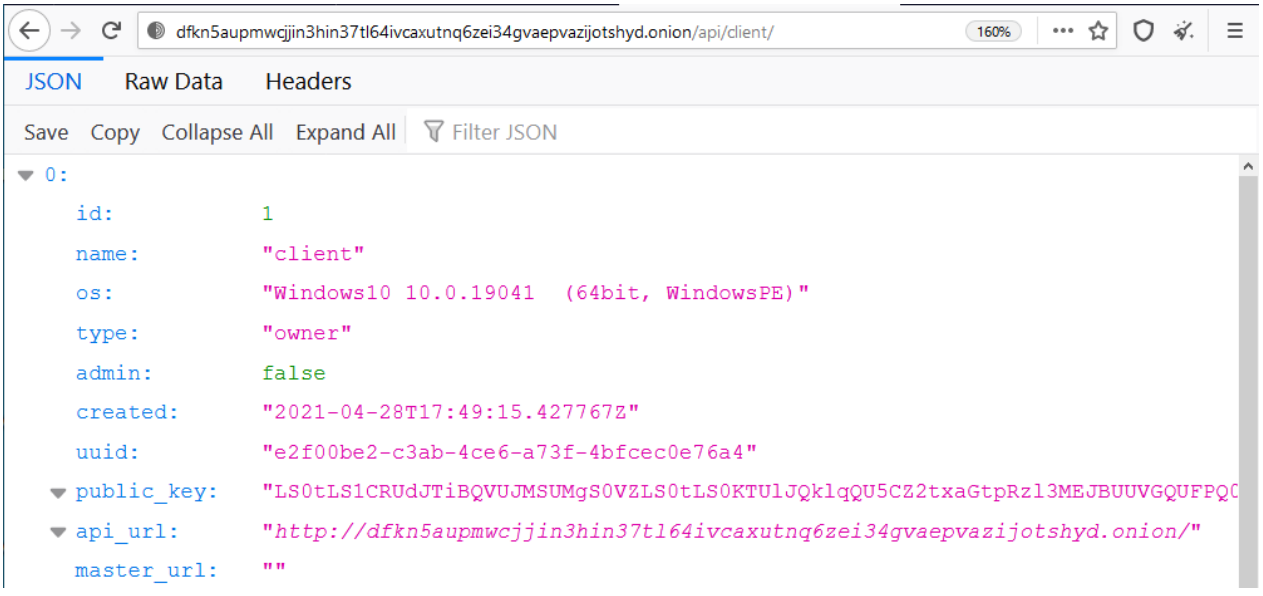
První počítač využíval adresu http://dfkn5...shyd.onion/. Tento počítač můžeme označit jako pacient nula. Neměl žádný master uzel, nicméně sloužil jako master uzel pro ostatní.

Nákaza u tohoto uzlu probíhala podobně jako u ostatních uzlů až na ten rozdíl, že tento uzel nebyl dále šířen do sítě tak, jak by to bylo v případě jiných uzlů, které mají k dispozici master uzel. Tím, že se do runneru přidá ještě API, se výsledná velikost runneru výrazně zvětší. Z tohoto důvodu, lze API stáhnout dodatečně od dalších uzlů, ale je to časově velmi náročné.

Při nákaze runner spustil Tor a API pomocí subprocessu. Tor se tímto způsobem spouštěl pokaždé. Při startu počítače se o spuštění runneru a API postaral operační systém, díky hodnotám uložených v registrech.



Obrázek 12.5: Nastavení Onion Service pro pacienta nula



Obrázek 12.6: První nakažený počítač, po spuštění runneru je dostupný přes Onion Service

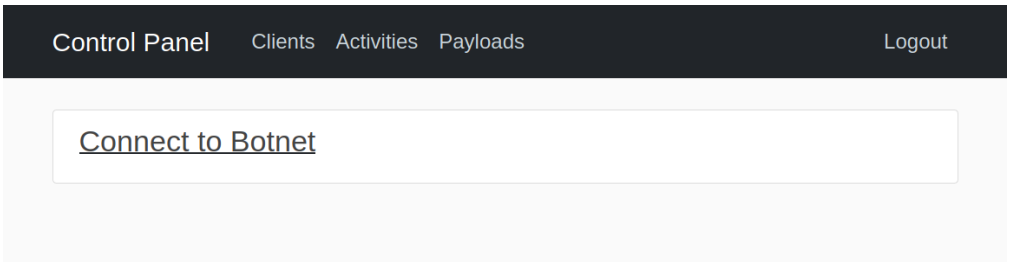
Po spuštění prvního uzlu jsem následně spouštěl další klienty. Dalším klientem byl uzel http://i2pyi...3hid.onion/, který měl jako master uzel http://dfkn5...shdy.onion/. Po registraci klienta http://i2pyi...3hid.onion/ u své lokální API je uzel distribuován uzlu http://dfkn5...shdy.onion/ a naopak http://dfkn5...shdy.onion/ předá informace o svých klientech do databáze uzlu http://i2pyi...3hid.onion/.


```
1:
    id:          2
    name:        "Client"
    os:          "Windows10 10.0.19041  (64bit, WindowsPE)"
    type:        "guest"
    admin:       false
    created:     "2021-04-28T17:54:28.889192Z"
    uuid:        "c1c4d0d8-154e-48a7-9c9f-ed4194f3d36e"
  public_key:    "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlJQklqQU5CZ2txaGtpRzl3MEJBUUVGQUFPQ
  api_url:       "http://i2pyi4lgnxsdsxexkvrym24wsmjmoq3b3pgccyuarczyq2j6kpnr3hid.onion/"
  master_url:    "http://dfkn5aupmwcjjin3hin37tl64ivcaxutnq6zei34gvaepvazijotshyd.onion/"
```


Obrázek 12.7: Druhý nakažený počítač

12.1.3 Konfigurace Control Center

V tuto chvíli přichází na řadu spuštění Control Centra. Control Center nepouštíme jako jeden spustitelný soubor, ale klasicky pomocí Pythonu. V Control Center musíme vytvořit jednoho falešného uživatele, abychom se tím připojili do botnetu a masternode nám předal veškeré záznamy o klientech. Nového klienta lze vytvořit klasickým způsobem, a to tak, že u sebe na počítači spustíme runner. Tento způsob je možný jen dokud nejsou v botnetu žádné škodlivé payloady. Lepším způsobem je využít možnosti Control Centra a nového klienta vytvořit přímo tam. Pokud nejsou v databázi žádní klienti Control Center nabídne možnost Connect to Botnet.



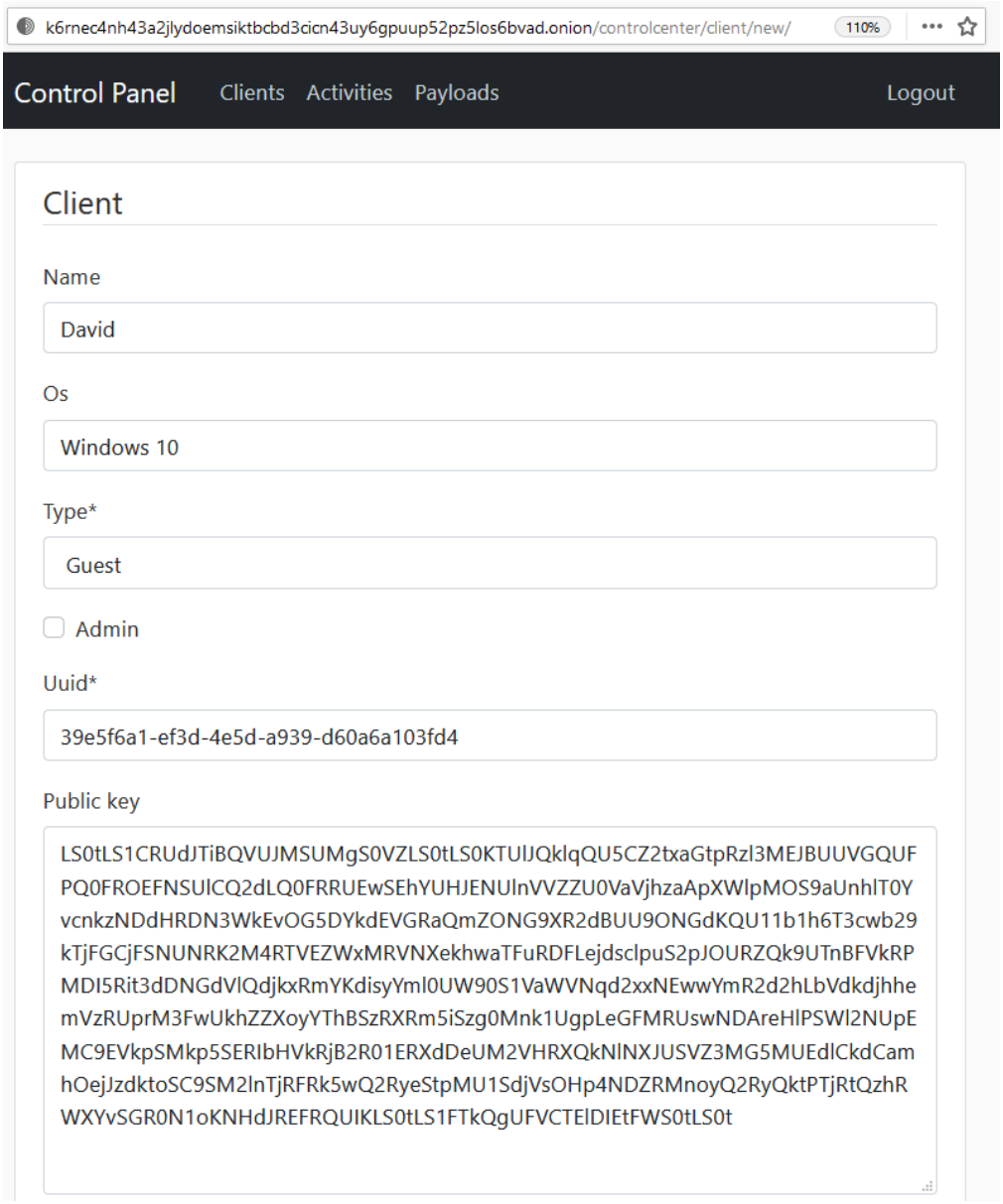
Obrázek 12.8: Odkaz pro vytvoření nového klienta

Po kliknutí na odkaz se zobrazí formulář, ve kterém je nutné vyplnit všechny důležité parametry. Před odesláním je ještě potřeba vygenerovat šifrovací klíče.

K tomu využijeme zdrojových kódů runnera a Python v interaktivním režimu. Pomocí Pythonu naimportujeme třídu Encryption, vytvoříme její novou instanci a zavoláme metodu generate_client_keys(). Následně naimportujeme knihovnu pro práci s base64 pomocí import base64 a veřejný klíč, který jsme si před chvílí vygenerovali pomocí base64, zakódujeme. Takto zakódovaný klíč zkopírujeme do vstupního pole v Control Center. Před stisknutím tlačítka Create Client ještě odešleme na naše API POST požadavek s cestou k privátnímu klíči. K tomu využijeme knihovnu requests.

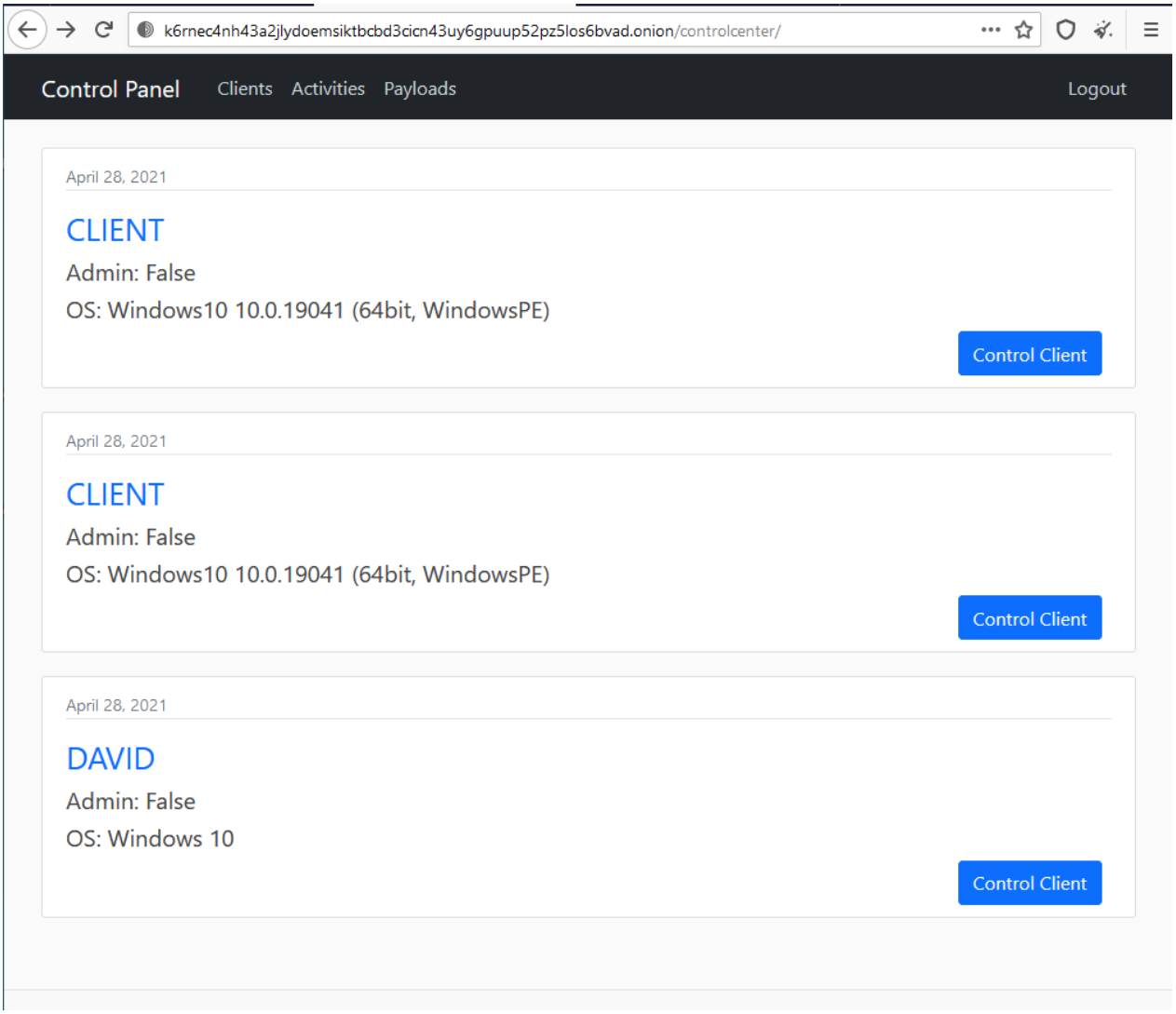
```
from encryption import Encryption
enc = Encryption()
enc.generate_client_keys()
requests.post('http://127.0.0.1:8000/api/encryption-init/', data = {"path": enc.
    private_pem_path})
import base64
base64.b64encode(enc.public_key_pem)
```

Listing 12.1: Inicializace šifrování



Obrázek 12.9: Formulář pro vytvoření nového klienta

V tuto chvíli můžeme formulář odeslat. Nový klient se vytvoří a současně se vyplní tabulka všemi klienty botnetu.



Obrázek 12.10: Master node nám předal veškeré informace o botnetu

12.1.4 Použití Control Center

Pro demonstraci můžeme využít payload PrintScreen. Ke spuštění payloadu potřebujeme vytvořit novou aktivitu. K tomu slouží tlačítko Control Client. Po výběru klienta tedy klikneme na Control Client a z nabídky vybereme Print Screen a potvrdíme vytvoření aktivity. Po vytvoření aktivity dojde k přesměrování uživatele na stránku s detailem aktivity s nápisem Request in Progress. Ve chvíli, kdy je požadavek splněn, se stránka automaticky obnoví a my uvidíme snímek obrazovky klientova počítače.

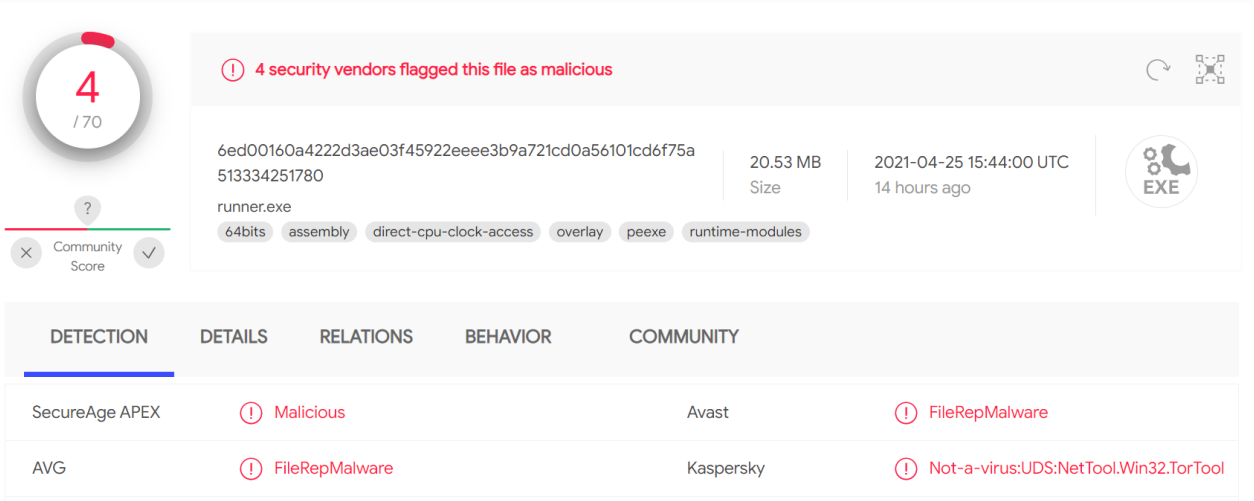
12.2 Zhodnocení prvního experimentu

Na základě výše uvedených skutečností lze konstatovat, že praktická část diplomové práce je funkční. V experimentu jsem prokázal funkcionalitu runneru, který byl schopen nakazit počítače s operačním systémem Windows 10 a následně nakažené počítače samostatně kontaktovaly vzdálené uzly, od kterých si vyžádaly payloadu, jenž následně vykonaly. Také byla prokázána další možnost zadávání příkazů nakaženým počítačům pomocí Control Center.

12.3 Experiment druhý: odolnost vůči antivirovým programům

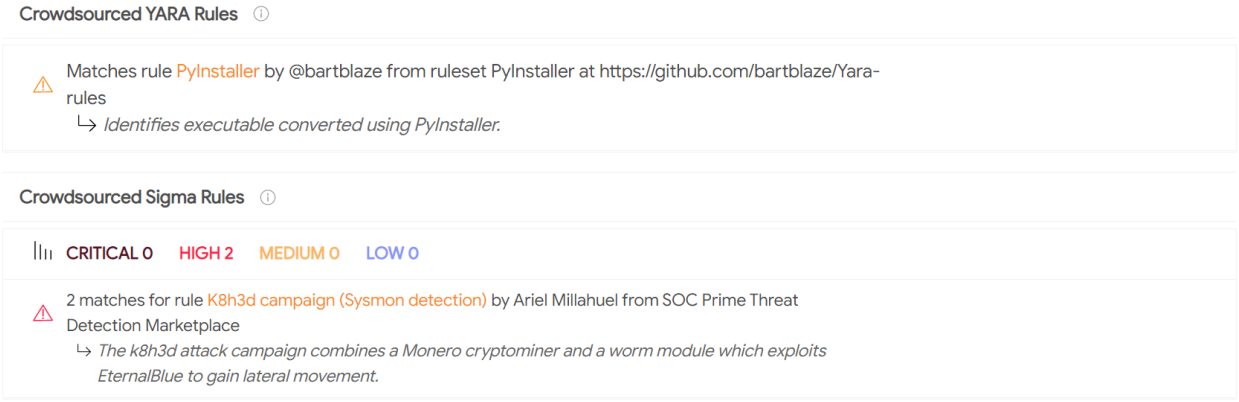
V tomto experimentu jsem otestoval odolnost malwaru vůči antivirovým řešením. Testoval jsem dva vzorky runneru. První vzorek neměl uvnitř zdrojového kódu definovanou onion adresu master uzlu. Tento vzorek jsem označil jako runner_nomaster.exe. Druhý runner měl adresu nastavenou na: "dfkn5...shyd.onion". Právě natvrdo zapsaná onion adresa mohla ovlivnit výsledek testu. Vzorek nesl označení runner.exe. Runnery byly vytvořeny pomocí Pythonu 3.9 a nejnovější verze PyInstalleru, nainstalovaného pomocí pip. Ty samé vzorky jsem následně ještě zkusil vytvořit pomocí PyInstalleru, který jsem sestavil přímo ze zdrojových kódů, které jsou dostupné na Githubu. Název těchto vzorků končí na "_compiled". O tomto způsobu obejití detekce jsem se dozvěděl na základě přečtení několika příspěvku na webu StackOverflow, kde uživatelé řešili falešně pozitivní nálezy antivirů u libovolného typů aplikací vytvořených pomocí nástroje PyInstaller. Testování probíhalo pomocí webové služby VirusTotal. [35] [63]

Testovaný vzorek nesl onion adresu uvnitř kódů. Vzorek označily 4 ze 64 antivirů za pozitivní. AVG a AVAST označily malware jako FileRepMalware, což vypadá spíše na automatické označení malwarů, které používají PyInstaller.



Obrázek 12.11: Analýza runner.exe pomocí VirusTotal

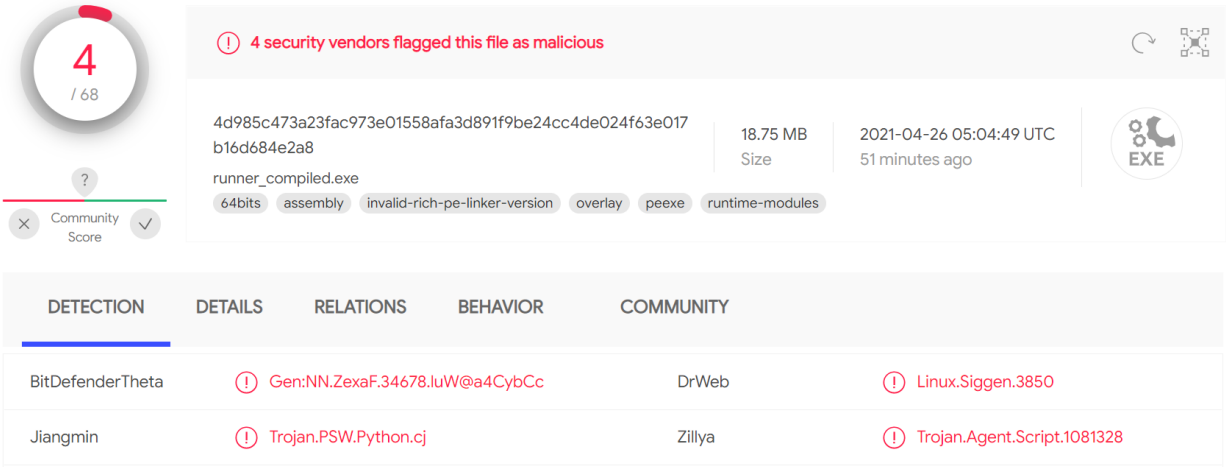
U tohoto i dalších měřeních byla zachycena níže uvedená Sigma a YARA pravidla. YARA pravidla zachycovala použití PyInstalleru. Pravidlo Sigma detekovalo specifický malware, který je však taktéž postavený na PyInstalleru.



Obrázek 12.12: Yara pravidla pro zachycená PyInstalleru

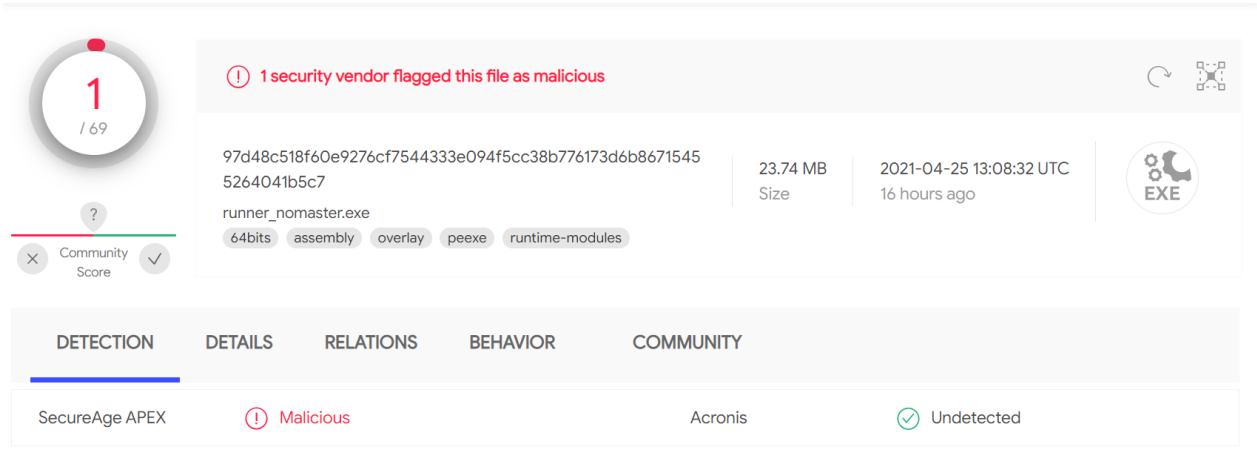
Kaspersky označil runner jako Not-a-virus:UDS:NetTool.Win32.TorTool. Pro tento runner jsem totiž využil archiv s Tor Expert Bundle stažený přímo z oficiálních stránek Tor Project. [64]

V případě toho samého vzorku, vytvořeného pomocí sestaveného Pyinstalleru, runner jako malware detekovalo 4 ze 68 antivirů. Ze seznamu vypadl AVAST a AVG. Kaspersky taktéž zůstal bez nálezu.



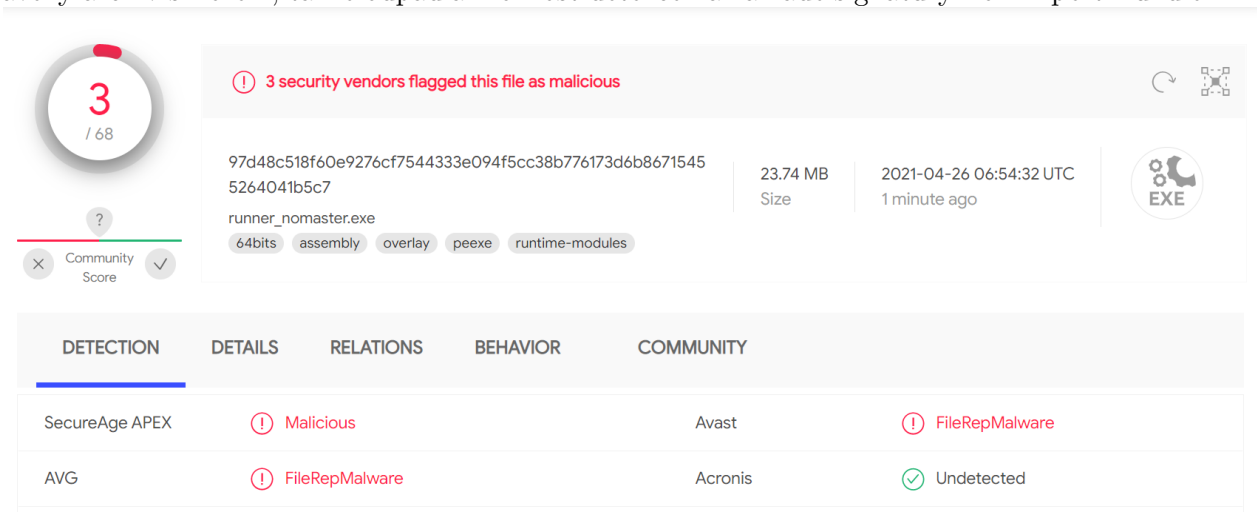
Vzorek runneru, který neměl specifikovanou onion adresu a byl vytvořen pomocí klasické instalace PyInstalleru, byl nejdříve označen jen jedním antivirem.

Obrázek 12.13: Analýza runneru_compiled.exe pomocí VirusTotal



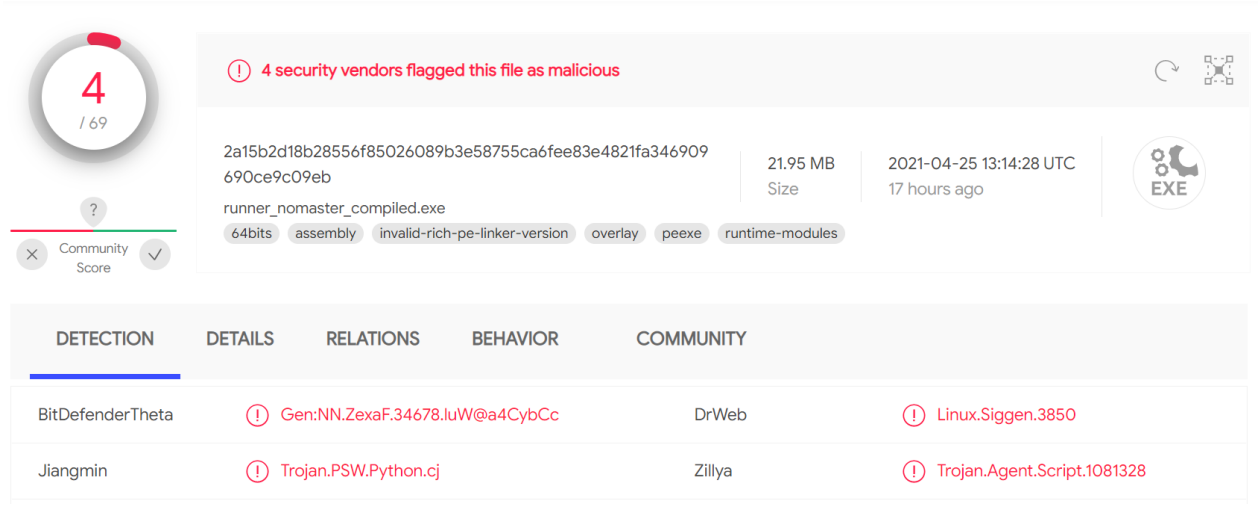
Obrázek 12.14: Analýza runner_nomaster.exe pomocí VirusTotal

Na základě opakovaného testování se již objevily nálezy i u dalších antivirů. Kromě SecureAge APEX se následně objevily AVG a Avast s označením FileRepMalware, tedy stejné označení jako u prvního testu. Ve výsledku tedy 3 pozitivní nálezy ze 68 dostupných antivirů. Tento vzorek měl upravený archiv s Torem, takže odpadla možnost detekce na základě signatury Tor Expert Bundle.



Obrázek 12.15: Opakovaná analýza runner_nomaster.exe pomocí VirusTotal

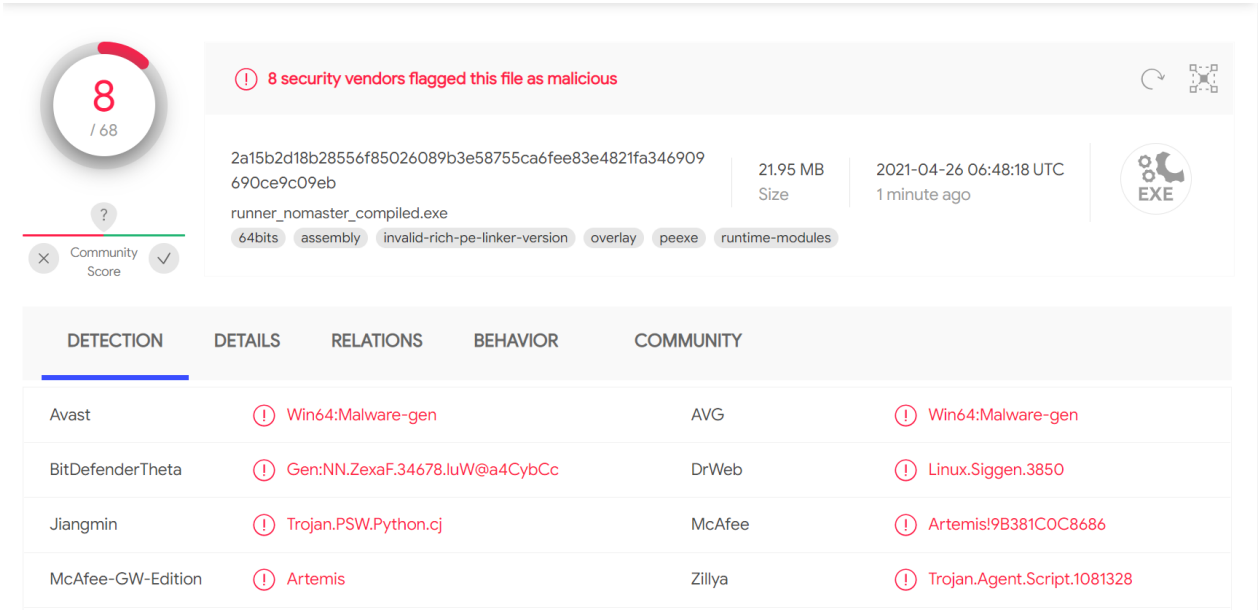
Jako poslední jsem otestoval vzorek bez onion master adresy vytvořený pomocí sestaveného PyInstalleru. Při prvním testu měl vzorek pouze 4 pozitivní detekce.



Obrázek 12.16: Analýza runner_nomaster_compiled.exe pomocí VirusTotal

Po opakovaném testování přibyl např. McAffe, který vzorek označil za Artemis. Slovo Artemis označuje ochranu v reálném čase a !9b381c0c868 potom začátek MD5 daného souboru, který označuje, že vzorek byl detekován již dříve McAffee real-time ochranou a byl nějakým způsobem detekován jako škodlivý. [65]

Win64:Malware-gen potom popisuje chování trojského koně, jako je stahování dodatečného softwaru, odesílání informací o počítači, popřípadě otevření zadních vrátek pro možnost vzdáleného přístupu útočníka do počítače. [66]



Obrázek 12.17: Opakovaná analýza runner_nomaster_compiled.exe pomocí VirusTotal

12.4 Shrnutí

Na základě výše popsaného experimentu můžeme říct, že je malware relativně odolný vůči detekci skenů antivirových řešení.

Vlastnoručně sestavený PyInstaller zároveň může ovlivnit výsledek především u antiviru AVG a Avast.

Jediné, co můžeme malware ohrozit, je dobrá real-time ochrana. To potvrzuje také jeden z posledních testu, kdy McAfee při prvním spuštění malware nedetekoval, nicméně při dalším testu již na základě real-time ochrany malware přesunul do karantény, a později již malware detekoval a označil jako Artemis.

Malware jsem také zkoušel otestovat na virtuálním počítači. Windows Defender taktéž nedetekoval malware při skenování souboru, ale až při jeho spuštění.

Ve chvíli, kdy jsem malware spustil na počítači se zapnutým Windows Defender a nainstalovanými nejnovějšími aktualizacemi, tak Windows Defender v některých případech byl schopen zastavit nebo alespoň velmi ovlivnit funkci malwaru. Nejčastější problém byl zablokovaný Tor, který na pozadí spouští Onion Service.

Většinou Windows Defender detekoval malware na základě zápisu do registru a při překopírovávání některých spustitelných souborů do složky AppData.

Kapitola 13

Zhodnocení

V rámci této diplomové práce jsem vytvořil koncept malwaru, který využívá některé prvky hejnové inteligence a zároveň komunikuje po síti Tor.

Hejnová inteligence je charakteristická tím, že v ní probíhá komunikace mezi členy hejna. V mém případě po připojení nového nakaženého počítače do botnetu je tomuto uzlu předán kompletní seznam již existujících klientů. Nový klient následně tyto uzly kontaktuje a vyžádá si od nich škodlivý kód, který následně vykoná. Zároveň tomuto uzlu můžeme na dálku předávat další příkazy, které má vykonávat formou aktivit. Každý takový klient u sebe provozuje Onion Service, díky které je dostupný komukoliv v síti Tor, kdo zná jeho onion adresu.

Na základě výše zmíněných informaci můžeme označit cíl práce za splněný.

Vzhledem k tomu, že jsem diplomovou práci zpracovával v době pandemie Covid-19, byl jsem omezen výpočetním výkonem svého počítače. V domácím prostředí nejsem schopen spustit více jak tři virtuální počítače. Na výkonnějším počítači by bylo možné experiment simulovat na více počítačích.

V reálném prostředí nedoporučuji malware testovat.

Kapitola 14

Závěr

V této práci jsem předvedl koncept hybridního malwaru, který využívá kombinaci principu hejnové inteligence a klasického botnetu s využitím sítě Tor a technologiích postavených na jazyce Python.

V teoretické práci jsem nastínil možnosti tvorby malwaru v jazyce Python a zmínil jsem základní stavební bloky, které jsou pro malware v tomto jazyce typické. Od způsobu spuštění Python kódu, přes funkce a balíčky, které v případě Pythonu můžeme použít, až po zabalení výsledného malwaru. Na závěr jsem přidal reálné ukázky malware vytvořené jazykem Python.

Dále jsem sepsal základní informace o síti Tor, zmínil software, který je nezbytný pro přístup do této sítě, a popsal základní myšlenku služeb Onion.

V poslední části jsem zmínil nejznámější algoritmy využívající hejnovou inteligenci a stanovil, kterými prvky těchto algoritmů se můj malware bude inspirovat.

V kapitole Návrh řešení jsem zmínil důvody, které mě vedly k volbě programovacího jazyka Python. Uvedl jsem, jakým způsobem lze řešit komunikaci přes síť Tor, a stanovil jsem, že pro komunikaci přes síť Tor využiji Tor Expert Bundle namísto knihovny v Pythonu.

V části implementace jsem blíže popsal komponenty, ze kterých se výsledné řešení skládá, a popsal způsob, jakým lze řešení dále rozšiřovat, ladit a testovat.

V poslední části jsem funkcionalitu řešení předvedl pomocí praktické ukázky a zároveň toto výsledné řešení zhodnotil.

Tato práce může sloužit pro studenty počítačové bezpečnosti, kteří se chtějí dozvědět nové informace o architektuře malwaru, která je schopná na dálku vykonávat škodlivý kód. Současně tato práce může sloužit jako důkaz, že malware se může šířit i bez centrálního uzlu. V praxi by se tato práce mohla využít v oblasti penetračního testování. Praktickou část této práce by bylo možné zkombinovat s již existujícími hacking nástroji.

Díky této práci je možné na podobné hrozby v budoucnu reagovat a zaměřit se na jednotlivé prvky, ze kterých se takový malware může skládat, což lze využít pro rychlejší odhalení takového malwaru v praxi.

Současně jsem touto prací chtěl dokázat, že je možné pomocí jednoho programovacího jazyka, v tomto případě Python, vytvořit komplexní malwarovou architekturu, což se mi potvrdilo.

Literatura

1. DIAKOPOULOUS, Nick; BAGAVANDAS, Mythili. *Interactive: The Top Programming Languages* [online]. IEEE Spectrum, 2020 [cit. 2020-12-15]. Dostupné z: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020.
2. AZARIA, Johnathan; NAKAR, Ori; EDI, Kogan. *The World's Most Popular Coding Language Happens to be Most Hackers' Weapon of Choice* [online]. 2018-09-26 [cit. 2020-12-15]. Dostupné z: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020.
3. FAKHAR, Imam. *Malware spotlight: Fileless malware* [online]. 2019-12-30 [cit. 2020-12-14]. Dostupné z: https://resources.infosecinstitute.com/topic/malware-spotlight-fileless-malware/.
4. SNOVER, Jeffrey. *PowerShell is open sourced and is available on Linux* [online]. 2016-08-18 [cit. 2020-12-14]. Dostupné z: https://azure.microsoft.com/en-us/blog/powershell-is-open-sourced-and-is-available-on-linux/.
5. ZHANG, Xiaopeng; NAVARRETE, Chris. *Microsoft Word File Spreads Malware Targeting Both Apple Mac OS X and Microsoft Windows* [online]. 2019-12-30 [cit. 2020-12-14]. Dostupné z: https://resources.infosecinstitute.com/topic/malware-spotlight-fileless-malware/.
6. HACQUEBORD, Feike. *The Mysterious Mevade Malware* [online]. 2013-09-05 [cit. 2020-12-14]. Dostupné z: https://blog.trendmicro.com/trendlabs-security-intelligence/the-mysterious-mevade-malware/.
7. SANTOS, Roddell. *Adware Spread Alongside Mevade Variants, Hits Japan and US* [online]. 2013-09-05 [cit. 2020-12-14]. Dostupné z: https://blog.trendmicro.com/trendlabs-security-intelligence/us-taiwan-most-affected-by-mevade-malware/.
8. PIERLUIGI, Paganini. *Malware in Dark Web* [online]. 2018-01-15 [cit. 2020-12-14]. Dostupné z: https://resources.infosecinstitute.com/topic/malware-dark-web/.

9. FAOU, Matthieu; CÔTÉ CYR, Alexandre. *KryptoCibule: The multitasking multicurrency cryptostealer* [online]. 2020-09-02 [cit. 2020-12-15]. Dostupné z: `https://www.welivesecurity.com/2020/09/02/kryptocibule-multitasking-multicurrency-cryptostealer/`.

10. SIKORA, Lubomír. *Swarm Malware* [online]. Ostrava, 2017 [cit. 2019-10-14]. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava. Dostupné z: `http://hdl.handle.net/10084/119183/`.

11. KOLOUCH, Jan. *CyberCrime*. Praha: CZ.NIC, 2016. ISBN 978-80-88168-18-8.

12. STROUKAL, Dominik. *Dark Web: sex, drogy a bitcoiny*. Praha: Grada, 2020. ISBN 978-80-271-2934-8.

13. *Glossary: Tor/Tor network/Core Tor* [online]. Tor Project, 2020-03-31 [cit. 2021-04-02]. Dostupné z: `https://support.torproject.org/glossary/tor-tor-network-core-tor/`.

14. *About Tor: Why is it called Tor?* [Online]. Tor Project, 2020-03-31 [cit. 2021-04-02]. Dostupné z: `https://support.torproject.org/about/#why-is-it-called-tor`.

15. *Glossary: Relay* [online]. Tor Project, 2020-03-31 [cit. 2021-04-02]. Dostupné z: `https://support.torproject.org/glossary/relay/`.

16. *Glossary: Circuit* [online]. Tor Project, 2020-03-31 [cit. 2021-04-02]. Dostupné z: `https://support.torproject.org/glossary/circuit/`.

17. *Glossary: Onion Services* [online]. Tor Project, 2020-03-31 [cit. 2021-04-02]. Dostupné z: `https://support.torproject.org/glossary/onion-services/`.

18. *Advanced configuration: HTTPS for your onion service* [online]. Tor Project, 2020-03-31 [cit. 2021-04-02]. Dostupné z: `https://community.torproject.org/onion-services/advanced/https/`.

19. BEN, Ben Wolford. *ProtonMail: Our Tor encrypted email site has a new SSL certificate* [online]. ProtonMail, 2018-03-27 [cit. 2021-04-02]. Dostupné z: `https://protonmail.com/blog/onion-ssl-certificate/`.

20. *Onion Services: How do onion services work?* [Online]. Tor Project, 2021-01-08 [cit. 2021-04-02]. Dostupné z: `https://community.torproject.org/onion-services/overview/`.

21. OPLATKOVÁ, Zuzana; OŠMERA, Pavel; ŠEDA, Miloš; VČELAŘ, František; ZELINKA, Ivan. *Evoluční výpočetní techniky: principy a aplikace*. BEN, 2008.

22. DORIGO, Marco. *Optimization, Learning and Natural Algorithms (in Italian)*. Milan, Italy, 1992. Dis. Dipartimento di Elettronica, Politecnico di Milano.

23. KENNEDY, James; EBERHART, Russell. Particle swarm optimization. In: *IEEE International Conference on Neural Networks - Conference Proceedings*. 1995, sv. 4, s. 1942–1948. Dostupné také z: `www.scopus.com`.

24. KRISHNANAND, K. N.; GHOSE, D. Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications. *Multiagent and Grid Systems*. 2006, roč. 2, č. 3, s. 209–222. Dostupné také z: www.scopus.com. Cited By :144.

25. KARABOGA, Dervis. An Idea Based on Honey Bee Swarm for Numerical Optimization, Technical Report - TR06. *Technical Report, Erciyes University*. 2005-01.

26. *About Cython* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://www.jython.org/.

27. *What is Jython?* [Online]. 2021 [cit. 2021-04-03]. Dostupné z: https://www.jython.org/.

28. *IronPython* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://ironpython.net/.

29. *IronPython 3.4.0-alpha1* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://github.com/IronLanguages/ironpython3/releases/tag/v3.4.0-alpha1.

30. *PyPy - Features* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://www.pypy.org/features.html.

31. *Brython* [online]. 2021-09-10 [cit. 2021-04-03]. Dostupné z: https://github.com/brython-dev/brython.

32. HAYEN, Kay. *Nuitka* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://nuitka.net/pages/overview.html.

33. *PyInstaller* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://www.pyinstaller.org/.

34. *PyInstaller* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://pypi.org/project/pyinstaller/.

35. *Program made with PyInstaller now seen as a Trojan Horse by AVG* [online]. 2017-05-04 [cit. 2021-04-02]. Dostupné z: https://stackoverflow.com/questions/43777106/program-made-with-pyinstaller-now-seen-as-a-trojan-horse-by-avg?noredirect=1&lq=1.

36. *py2exe* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://www.py2exe.org/.

37. SCHOENTGEN, Mickaël. *Requests: HTTP for Humans* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://python-mss.readthedocs.io/index.html.

38. JACKSON, Austin. *Python Malware On The Rise* [online]. Cyborg Security, 2020-07-14 [cit. 2021-04-03]. Dostupné z: https://www.cyborgsecurity.com/cyborg_labs/python-malware-on-the-rise/.

39. REITZ, Kenneth. *Requests: HTTP for Humans* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://docs.python-requests.org/en/master/.

40. HAMMOND, Mark. *pywin32* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://pypi.org/project/pywin32/.

41. *Malware writing - Python Malware, part 1* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://0x00sec.org/t/malware-writing-python-malware-part-1/11700.

42. *PyCrypto* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://pypi.org/project/pycrypto/.

43. *PyCryptoDome* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://www.pycryptodome.org/.

44. *Built-in Functions: exec* [online]. Python Software Foundation, 2021, 2021 [cit. 2021-04-03]. Dostupné z: https://docs.python.org/3/library/functions.html#exec.

45. *Built-in Functions: eval* [online]. Python Software Foundation, 2021, 2021 [cit. 2021-04-03]. Dostupné z: https://docs.python.org/3/library/functions.html#eval.

46. *subprocess — Subprocess management* [online]. Python Software Foundation, 2017, 2021 [cit. 2021-04-03]. Dostupné z: https://docs.python.org/3/library/functions.html#eval.

47. *PyArmor* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://github.com/dashingsoft/pyarmor.

48. KOVACS, Eduard. *OnionDuke APT Malware Distributed Via Malicious Tor Exit Node* [online]. Security Week, 2014-11-14 [cit. 2021-04-03]. Dostupné z: https://www.securityweek.com/onionduke-apt-malware-distributed-malicious-tor-exit-node.

49. KOVACS, Eduard. *APT Group Uses Seaduke Trojan to Steal Data From High-Value Targets* [online]. Security Week, 2015-06-13 [cit. 2021-04-03]. Dostupné z: https://www.securityweek.com/apt-group-uses-seaduke-trojan-steal-data-high-value-targets.

50. GRUNZWEIG, Josh. *Python-Based PWOBot Targets European Organizations* [online]. Palo-alto Networks UNIT 42, 2016-04-14 [cit. 2021-04-03]. Dostupné z: https://unit42.paloaltonetworks.com/unit42-python-based-pwobot-targets-european-organizations/.

51. KENEFICK, Ian; YAMBAO, Mary; NIETO, Alvin; ANG, Kerr. *A Closer Look at the Locky Poser, PyLocky Ransomware* [online]. Trend Micro, 2018-09-10 [cit. 2021-04-03]. Dostupné z: https://www.trendmicro.com/en_us/research/18/i/a-closer-look-at-the-locky-poser-pylocky-ransomware.html.

52. BAUTISTA, Mike. *Pylocky Unlocked: Cisco Talos releases PyLocky ransomware decryptor* [online]. Cisco Talos Intelligence, 2019-01-10 [cit. 2021-04-03]. Dostupné z: https://blog.talosintelligence.com/2019/01/pylocky-unlocked-cisco-talos-releases.html.

53. MERCER, Warren; RASCAGNERES, Paul; VENTURA, Vitor. *PoetRAT: Python RAT uses COVID-19 lures to target Azerbaijan public and private sectors* [online]. Cisco Talos Intelligence, 2020-04-16 [cit. 2021-04-03]. Dostupné z: https://blog.talosintelligence.com/2020/04/poetrat-covid-19-lures.html.

54. MERCER, Warren; RASCAGNERES, Paul; VENTURA, Vitor. *PoetRAT: Malware targeting public and private sector in Azerbaijan evolves* [online]. Cisco Talos Intelligence, 2020-10-06 [cit. 2021-04-03]. Dostupné z: https://blog.talosintelligence.com/2020/10/poetrat-update.html.

55. *uncompyle6* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://pypi.org/project/uncompyle6/.

56. *Why Django?* [Online]. Django Software Foundation, 2021, 2021 [cit. 2021-04-03]. Dostupné z: https://www.djangoproject.com/start/overview/.

57. *Django REST framework* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://www.django-rest-framework.org/.

58. *Torpy* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://github.com/torpyorg/torpy.

59. *Waitress decryptor* [online]. Pylos Project, 2021-03-07 [cit. 2021-04-03]. Dostupné z: https://docs.pylonsproject.org/projects/waitress/en/stable/.

60. *logging — Logging facility for Python* [online]. Python Software Foundation, 2017, 2021 [cit. 2021-04-03]. Dostupné z: https://docs.python.org/3/library/logging.html.

61. BRANDL, George. *Sphinx: Python Documentation Generator* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://www.sphinx-doc.org/.

62. *The Django admin documentation generator* [online]. Django Software Foundation, 2021, 2021 [cit. 2021-04-03]. Dostupné z: https://docs.djangoproject.com/en/3.1/ref/contrib/admin/admindocs/#module-django.contrib.admindocs.

63. *PyInstaller: Building the Bootloader* [online]. 2021 [cit. 2021-04-03]. Dostupné z: hhttps://pyinstaller.readthedocs.io/en/stable/bootloader-building.html.

64. *Windows Expert Bundle* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://www.torproject.org/download/tor/.

65. *Some of my virus detections are called 'Artemis'* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://service.mcafee.com/webcenter/portal/cp/home/articleview?articleId=TS100414.

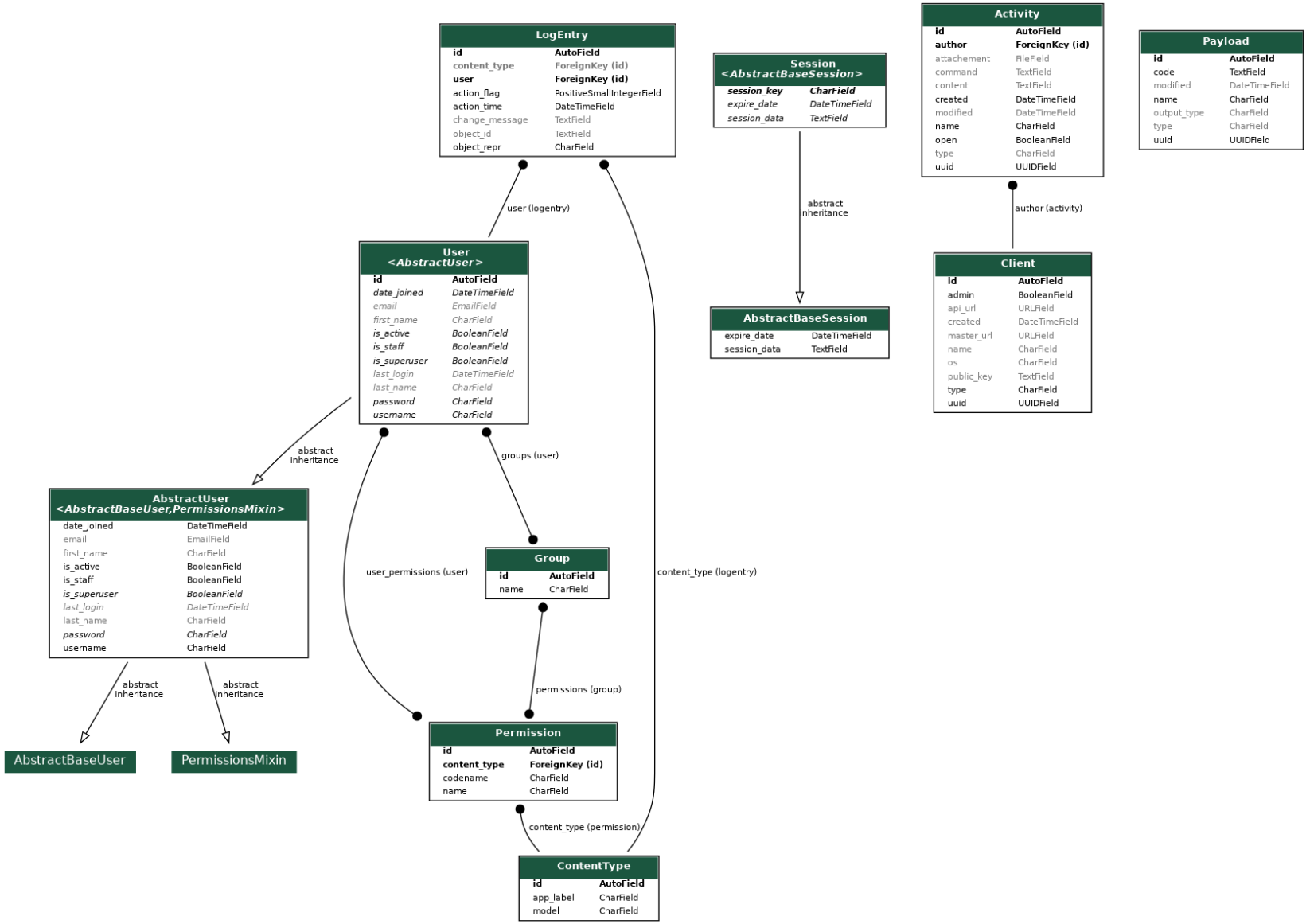
66. STELIAN, Pilici. *How to remove Win64:Malware-Gen Trojan (Virus Removal Guide)* [online]. 2017-09-25 [cit. 2021-04-03]. Dostupné z: https://malwaretips.com/blogs/remove-win64-malware-gen/.

Příloha A

Příloha v IS EDISON

Součástí diplomové práce je elektronická příloha v zip souboru.

- **ControlCenter** - Složka obsahuje zdrojové kódy pro Control Center.
- **Api** - Složka obsahuje zdrojové kódy pro API.
- **Runner** - Složka obsahuje zdrojové kódy pro runner.
- **Vzorky** - Složka obsahuje vzorky analyzované v druhém experimentu.
- **video.txt** - Soubor obsahuje odkaz na YouTube playlist s praktickými ukázkami.



Obrázek A.1: Databázový model (společný pro Control Center a API)